Approfondissement Docker

Docker et Kubernetes

CCI Strasbourg

Chapitre 3 : Approfondissement Docker

Objectifs

En suivant ce chapitre, nous allons approfondir les notions de Docker avec les modules suivants :

- 3.1 Réseau dans Docker
- 3.2 Gestion des volumes et persistance des données
- 3.3 Bonnes pratiques pour écrire des Dockerfiles
- 3.4 Sécurisation des conteneurs

Chapitre 3 : Approfondissement Docker

Module 3.1
Réseau dans Docker

Chapitre 3 : Approfondissement Docker

Module 3.1 – Réseau dans Docker

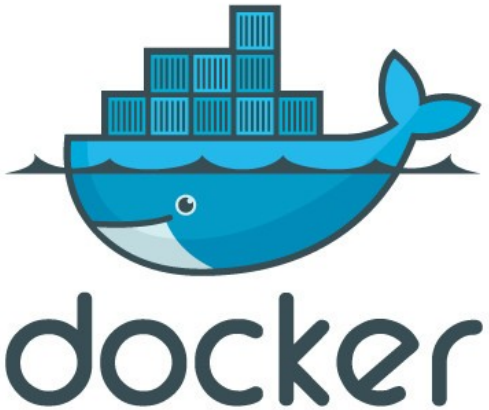
Sommaire :

- Présentation des réseaux dans Docker
- Les commandes de bases
- Les réseaux dans Docker Compose
- Manipulation des réseaux (TP)

Chapitre 3 : Approfondissement Docker

Module 3.1 – Réseau dans Docker

Présentation des réseaux dans Docker



Chapitre 3 : Approfondissement Docker

Module 3.1 – Présentation des réseaux dans Docker

Pour rappel, un réseau permet la communication entre deux équipements.

Au niveau de Docker, la notion de réseau va permettre de définir la communication entre :

- Les conteneurs entre eux
- Les conteneurs et l'ordinateur hôte
- Les conteneurs et le « monde extérieur »

Chapitre 3 : Approfondissement Docker

Module 3.1 – Présentation des réseaux dans Docker

Les différents types de réseaux

Dans Docker, on retrouve plusieurs types de réseaux :

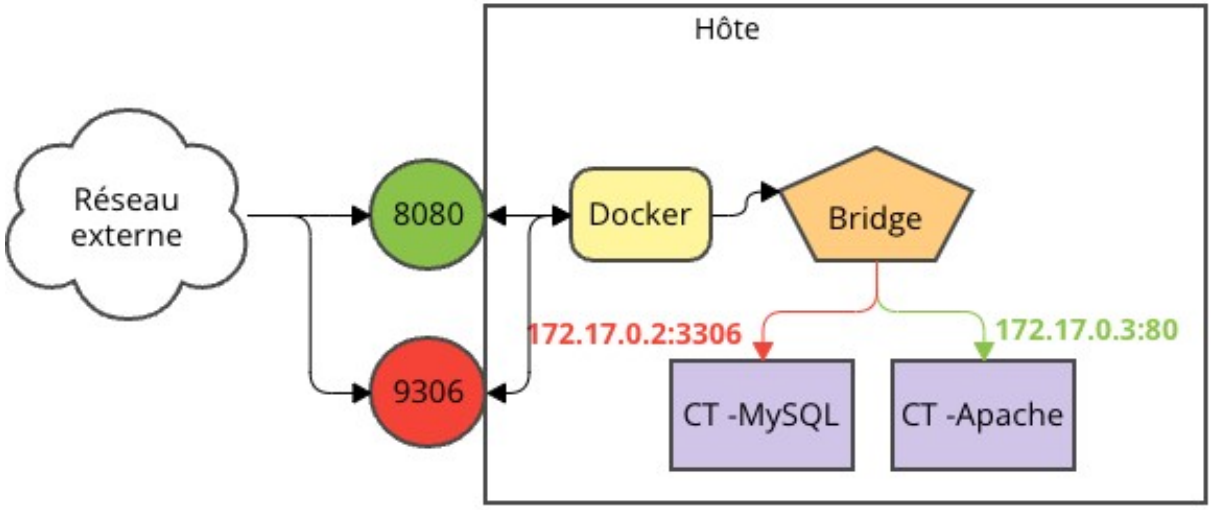
- **Bridge Network** : Le réseau par défaut pour les conteneurs. Les conteneurs connectés à ce réseau peuvent communiquer entre eux. Il s'agit du **type de réseau le plus couramment utilisé**.
- **None Network** : Interdit toute communication
- **Host Network** : Les conteneurs partagent le même espace réseau que l'hôte. Ils peuvent utiliser les ports de l'hôte directement.
- **Overlay Network** : Permet aux conteneurs de communiquer sur plusieurs hôtes Docker. Utile pour les applications distribuées.
- **Macvlan Network** : Donne aux conteneurs une adresse IP externe sur le réseau physique.

Chapitre 3 : Approfondissement Docker

Module 3.1 – Présentation des réseaux dans Docker

Le Bridge Network

Dans un réseau bridge, chaque conteneur connecté possède une interface réseau virtuelle distincte et le réseau va automatiquement attribuer une adresse IP à partir du pool disponible du réseau bridge. Un réseau nommé **docker0** est crée lors de l'installation de Docker, il sera utilisé par défaut pour connecter les conteneurs.



Chapitre 3 : Approfondissement Docker

Module 3.1 – Présentation des réseaux dans Docker

None Network

Le type de réseau None interdit toutes les communications avec le monde extérieur et les autres conteneurs. Il n’aura pas d’adresse IP et aura uniquement une interface locale de loopback.

Ce type de réseau est idéal si vous avez besoin de lancer un batch par exemple sur un conteneur et qui ne nécessite aucune dépendance avec d’autres services.

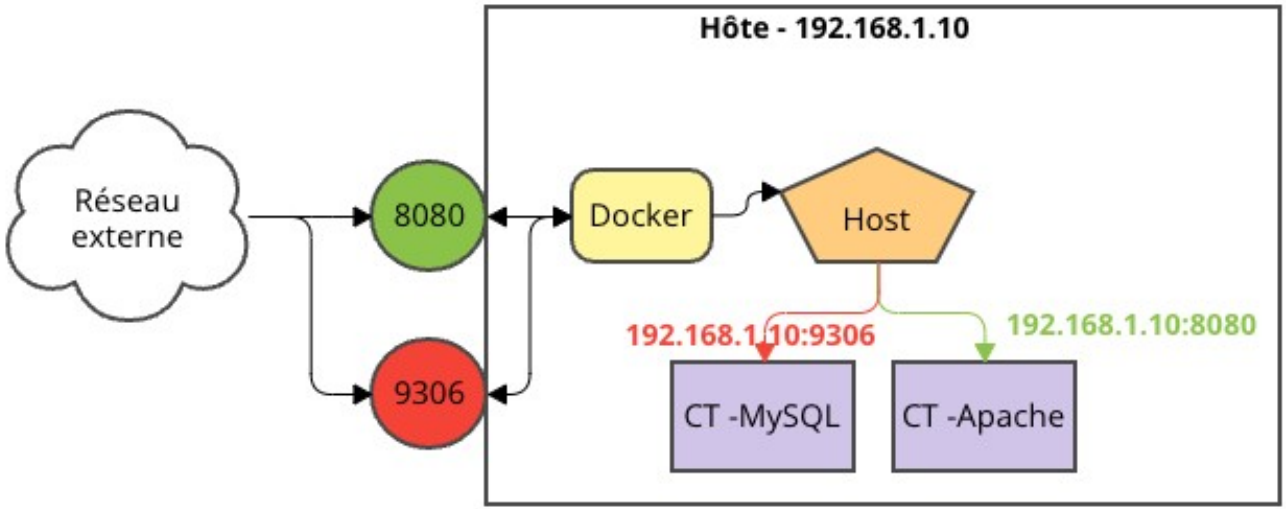
Chapitre 3 : Approfondissement Docker

Module 3.1 – Présentation des réseaux dans Docker

Host Network

Dans ce type de réseau, il n'y a plus d'isolation avec le monde extérieur, ce qui implique que :

- Les conteneurs auront la même adresse IP que l'hôte
- Les conteneurs vont directement utilisés les ports de l'hôte

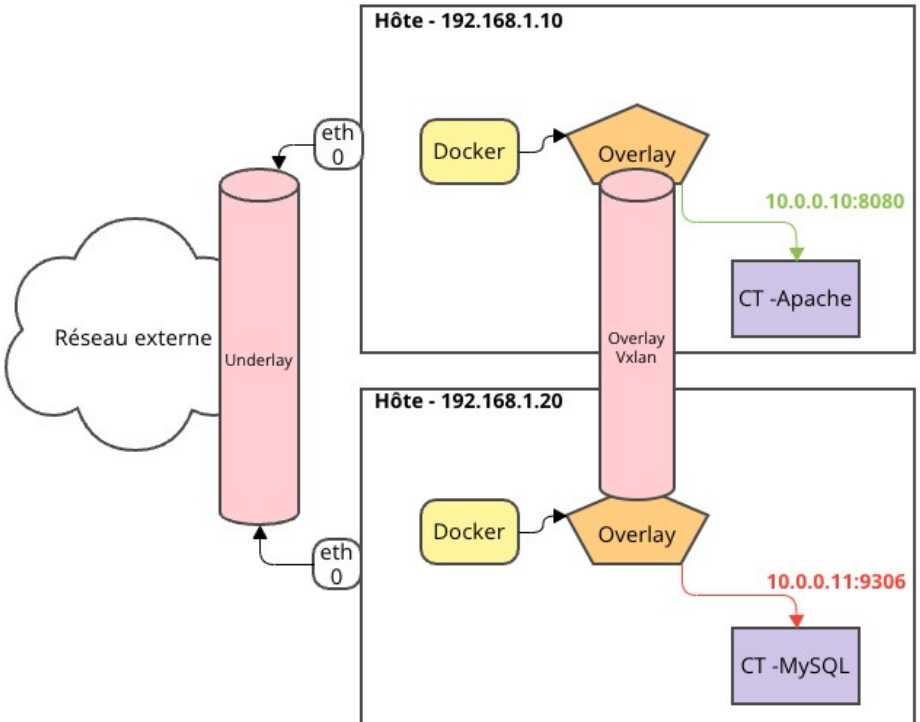


Chapitre 3 : Approfondissement Docker

Module 3.1 – Présentation des réseaux dans Docker

Overlay Network

L'overlay permet de connecter plusieurs hôtes Docker et d'effectuer une mise en commun de leurs réseaux.

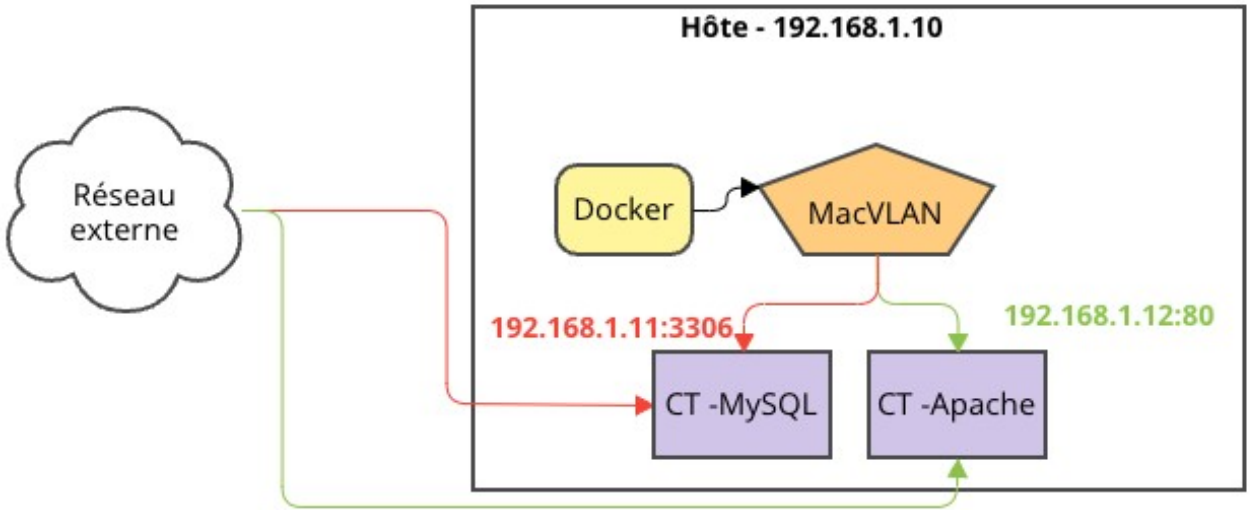


Chapitre 3 : Approfondissement Docker

Module 3.1 – Présentation des réseaux dans Docker

Macvlan Network

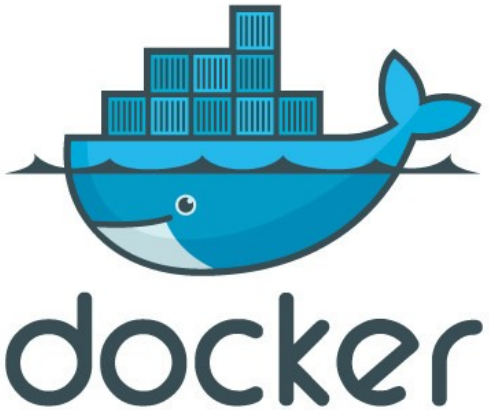
Macvlan permet d'assigner des adresses MAC aux conteneurs, ce qui permet de communiquer directement avec le réseau externe. Contrairement au réseau bridge, ici des adresses IPs différentes sont utilisées pour accéder aux conteneurs.



Chapitre 3 : Approfondissement Docker

Module 3.1 – Réseau dans Docker

Les commandes de base



Chapitre 3 : Approfondissement Docker

Module 3.1 – Les commandes de base

Gestion des réseaux

docker network create <nomReseau> : créer un nouveau réseau.
docker network create <nomReseau> : supprimer un réseau.
docker network ls : liste les différents réseaux
docker network inspect <nomReseau> : Inspecte le réseau

Chapitre 3 : Approfondissement Docker

Module 3.1 – Les commandes de base

Exemple création d'un réseau bridge « localnet »

docker network create --driver=bridge --subnet=172.16.0.0/16 --ip-range=172.16.1.0/24 --gateway=172.16.1.254 localnet

La commande ci-dessus va :

- créer un réseau nommé « localnet » en mode bridge (précisé avec l'option driver)
- déclarer un sous-réseau 172.16.1.0/16
- indiquer que l'attribution des adresses ip sera sur la plage 172.16.1.0/24
- que la passerelle pour sortir du réseau est l'adresse 172.16.1.254 (correspond au host)

Pour vous aider dans la déclaration des réseaux :

- https://www.ipaddressguide.com/cidr
- https://www.calculator.net/ip-subnet-calculator.html

Chapitre 3 : Approfondissement Docker

Module 3.1 – Les commandes de base

Utilisation avec les conteneurs

docker run --network=localnet -dit ubuntu bash
docker run --network=localnet -dit ubuntu bash
docker run --network=localnet --ip 172.16.1.20 -dit ubuntu bash

La commande ci-dessus va :

- Exécuter 3 conteneurs en arrière-plan sur le réseau bridge
  - Les deux premiers vont avoir une adresse IP automatique, le troisième sera mis manuellement par la commande

Pour vous aider dans la déclaration des réseaux :

- https://www.ipaddressguide.com/cidr
- https://www.calculator.net/ip-subnet-calculator.html

Chapitre 3 : Approfondissement Docker

Module 3.1 – Les commandes de base

Utilisation avec les conteneurs

docker network connect <nomReseau> <nomConteneur> : permet de connecter un conteneur à un réseau
docker network disconnect <nomReseau> <nomConteneur> : permet de déconnecter un conteneur d'un réseau

Remarques :

- pour connecter un conteneur à un réseau, le conteneur doit être actif
- il est possible de connecter un conteneur à plusieurs réseaux.

Chapitre 3 : Approfondissement Docker

Module 3.1 – Les commandes de base

Inspecter le réseau « localnet »

docker network inspect localnet

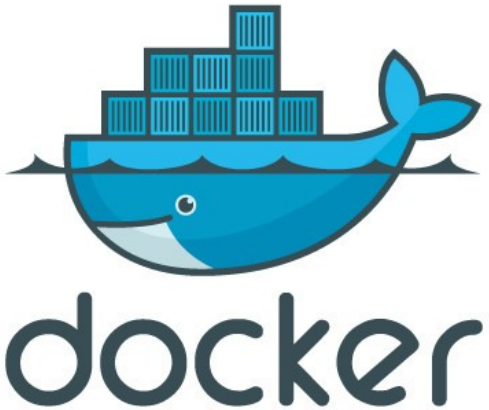

```
"Containers": {
    "a898efa9e1e093e89eb1f805150c84ecdd287fca1993cf
        "Name": "hopeful_rhodes",
        "EndpointID": "5a2582ca548576377aa9308d9d35
        "MacAddress": "02:42:ac:10:01:00",
        "IPv4Address": "172.16.1.0/16",
        "IPv6Address": ""
    },
    "c5226c2846b227d855efa5e5332338f28d6511b026200d
        "Name": "amazing_cerf",
        "EndpointID": "927ffba605ed146a16fbfe87d15c
        "MacAddress": "02:42:ac:10:01:14",
        "IPv4Address": "172.16.1.20/16",
        "IPv6Address": ""
    },
    "db630746260ce39e373ed8629b410334c8a429c86e720d
        "Name": "optimistic_dirac",
        "EndpointID": "c40f1c8c9ba05776a757ce681dcc
        "MacAddress": "02:42:ac:10:01:01",
        "IPv4Address": "172.16.1.1/16",
        "IPv6Address": ""
    }
},
```


On retrouve bien nos 3 conteneurs dans le réseau

Chapitre 3 : Approfondissement Docker

Module 3.1 – Réseau dans Docker

Les réseaux dans Docker Compose



Chapitre 3 : Approfondissement Docker

Module 3.1 – Les réseaux dans Docker Compose

Exemple de Docker compose

Docker Compose permet également de définir la gestion du réseau, ci-dessous une configuration de 2 conteneurs nginx sur un réseau bridge :

```
version: '3'

services:
  nginx_service_1:
    image: nginx
    networks:
      - my_bridge_network
    ports:
      - "8081:80"  # Mappe le port 80 du conteneur sur le port 8081 de l'hôte

  nginx_service_2:
    image: nginx
    networks:
      - my_bridge_network
    ports:
      - "8082:80"  # Mappe le port 80 du conteneur sur le port 8082 de l'hôte

networks:
  my_bridge_network:
    driver: bridge
    ipam:
      driver: default
      config:
        - subnet: 192.168.50.0/24
```

Chapitre 3 : Approfondissement Docker

Module 3.1 – Les réseaux dans Docker Compose

Exemple de Docker compose

services :

nginx_service_1 et nginx_service_2: Deux services Nginx utilisant l'image Nginx officielle.

networks: Spécifie le réseau auquel le service est connecté.

networks : Définit les réseaux utilisés par les services.

my_bridge_network : Le réseau de type Bridge avec la plage IP spécifiée.

driver : bridge : Indique que le réseau doit être de type Bridge.

ipam : Gère l'attribution des adresses IP.

driver : default: Utilise le driver d'attribution d'IP par défaut.

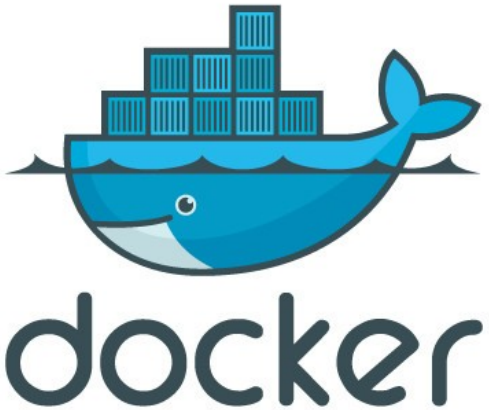
config : Configure le sous-réseau IP.

subnet : 172.16.50.0/24: Spécifie la plage IP pour le réseau.

Chapitre 3 : Approfondissement Docker

Module 3.1 – Réseau dans Docker

Manipulation des réseaux (TP)



Chapitre 3 : Approfondissement Docker

Module 3.1 – Manipulation des réseaux

Manipulation des réseaux (TP)

Suivre le TP fournit par le formateur.

Chapitre 3 : Approfondissement Docker

Module 3.1 – Questions ?

Chapitre 3 : Approfondissement Docker

Module 3.2
Gestion des volumes et persistance des données

Chapitre 3 : Approfondissement Docker

Module 3.2 – Gestion des volumes et persistance des données

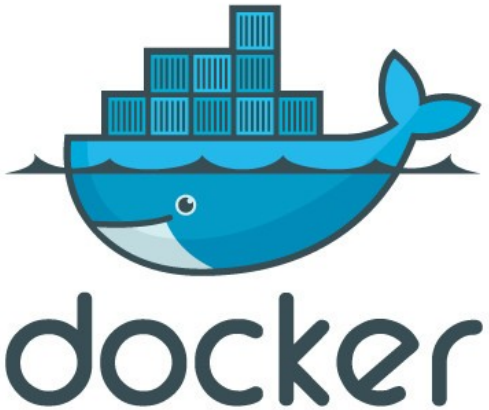
Sommaire :

- La notion des volumes dans Docker
- Les commandes de base sur les volumes
- Les volumes dans Docker Compose
- Manipulation des volumes (TP)

Chapitre 3 : Approfondissement Docker

Module 3.2 – Gestion des volumes et persistance des données

La notion des volumes dans Docker



Chapitre 3 : Approfondissement Docker

Module 3.2 – La notion des volumes dans Docker

Définition

Les volumes sont des mécanismes permettant de persister et de partager des données entre les conteneurs et l'hôte.

Sans cette possibilité, un conteneur ne pourrait pas être reprendre son état, en effet, les données à l'intérieur d'un conteneur sont éphémères.

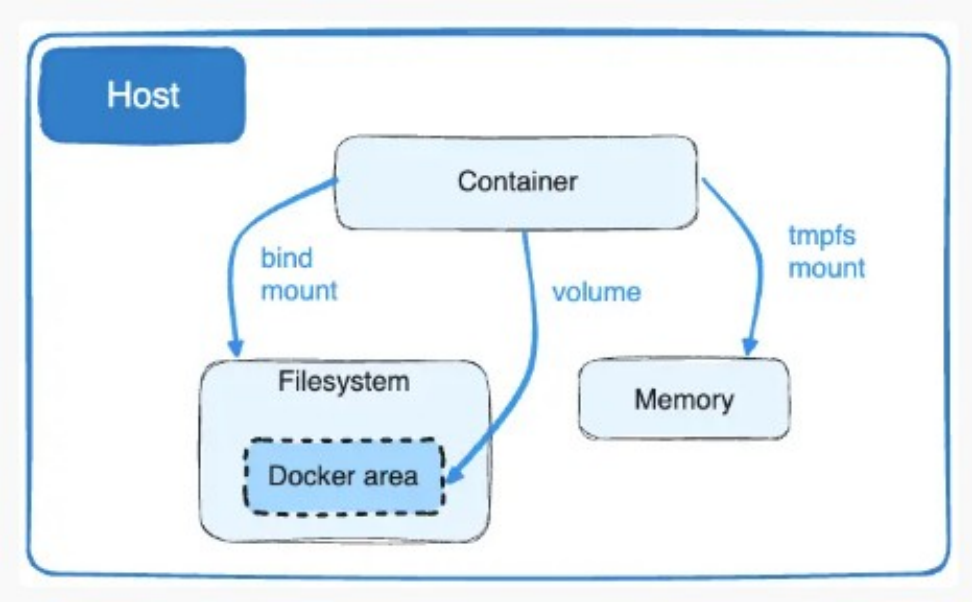
Les volumes offrent une solution pour stocker des données de manière persistante, même si le conteneur est supprimé.

Chapitre 3 : Approfondissement Docker

Module 3.2 – La notion des volumes dans Docker

Définition

Le schéma issu de la documentation officielle de Docker illustre ce propos :



Chapitre 3 : Approfondissement Docker

Module 3.2 – La notion des volumes dans Docker

Types de volumes

Au sein de Docker, on peut retrouver 3 types de volumes :

- **volume nommé :** il s'agit de volume déclaré et géré par Docker. Ces volumes peuvent être partagés par plusieurs conteneurs. La gestion de ces volumes passent par la commande **docker volume.**

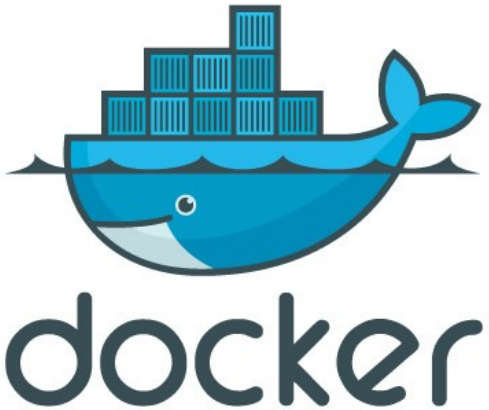
- **volume anonyme :** ce type de volume est crée par Docker lors de la création d'un conteneur quand l'option -v est passé dans la commande docker run. Ce type de volume est lié au conteneur et **ne peut pas être partagé**.

- **volume hôte :** un volume hôte est un fichier ou répertoire de l'hôte qui est relié au conteneur. La déclaration de ce type de volume est réalisé par l'option -v en précisant le chemin de l'hôte.

Chapitre 3 : Approfondissement Docker

Module 3.2 – Gestion des volumes et persistance des données

Les commandes de base sur les volumes



Chapitre 3 : Approfondissement Docker

Module 3.2 – Les commandes de base sur les volumes

Gestion des volumes

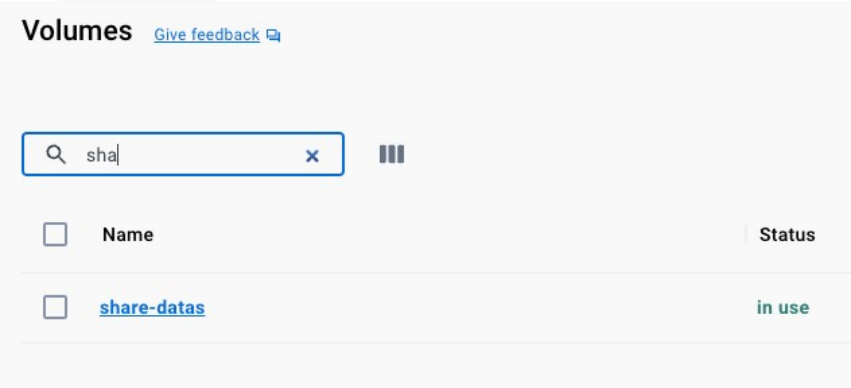
- **docker volume create <volume>** : création d'un nouveau volume nommé.
- **docker volume rm <volume>** : suppression d'un volume nommé.
- **docker volume ls** : liste les volumes.
- **docker volume inspect <volume>** : affiche les informations d'un volume.
- **docker run -v <cheminHote>:<cheminConteneur>** : monte un chemin hôte dans un conteneur.
- **docker run -v <cheminConteneur>** : création d'un volume anonyme pour le conteneur
- **docker run –mount source=<volume>,target=<cheminConteneur>**: monte un volume nommé dans un conteneur.

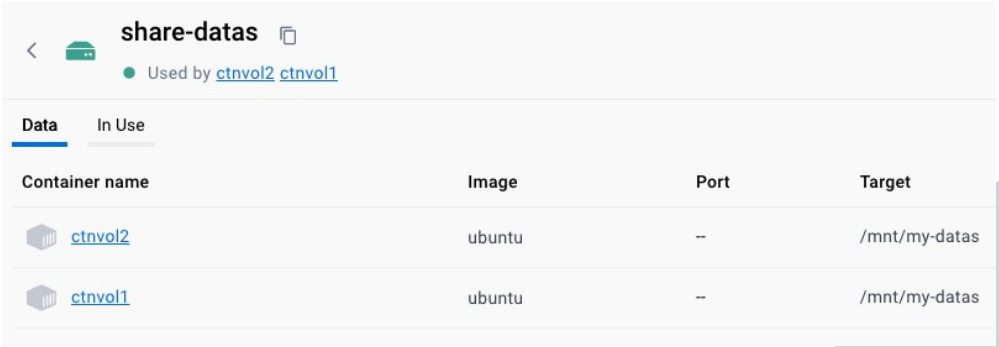
Chapitre 3 : Approfondissement Docker

Module 3.2 – Les commandes de base sur les volumes

Exemple avec Docker cli

- Création d'un volume et affectation à deux conteneurs :
  - docker volume create share-datas
  - docker run --name ctnvol1--mount source=share-datas,target=/mnt/my-datas -dit ubuntu bin/bash
  - docker run --name ctnvol2 --mount source=share-datas,target=/mnt/my-datas2 -dit ubuntu bin/bash





Chapitre 3 : Approfondissement Docker

Module 3.2 – Les commandes de base sur les volumes

Exemple avec Docker cli

- Création d'un conteneur avec un volume anonyme :
  - docker run --name docker-vol-anonyme -v /srv/test -dit ubuntu bin/bash

- Création d'un conteneur avec un volume hôte :
  - docker run --name docker-vol-hote -v /var/tmp/test:/srv/test -dit ubuntu bin/bash

Chapitre 3 : Approfondissement Docker

Module 3.2 – Les commandes de base sur les volumes

Exemple avec Docker cli

- Inspection d'un volume
  - **docker volume inspect share-datas**

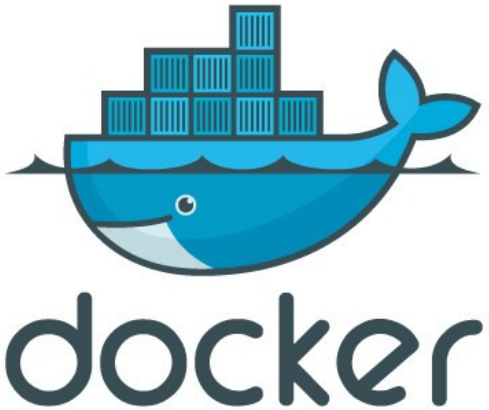
```
[
    {
        "CreatedAt": "2023-12-01T14:51:18Z",
        "Driver": "local",
        "Labels": null,
        "Mountpoint": "/var/lib/docker/volumes/share-datas/_data",
        "Name": "share-datas",
        "Options": null,
        "Scope": "local"
    }
]
```

- **Mountpoint** : chemin sur le host où se trouve les données du volume en clair
  Attention, sur MacOS, docker s'exécute via une vm et ce chemin n'existe pas, si vous voulez un shell de la vm, on peut utiliser la commande suivante :
  docker run -it --rm --privileged --pid=host justincormack/nsenter1

Chapitre 3 : Approfondissement Docker

Module 3.2 – Gestion des volumes et persistance des données

Les volumes dans Docker Compose



Chapitre 3 : Approfondissement Docker

Module 3.2 – Les volumes dans Docker Compose

Exemple de Docker compose

Docker Compose permet également de définir la gestion des volumes, ci-dessous un exemple de création d'un volume nommé « nginx-volume » qui est partagé entre deux conteneurs. Une modification le volume apporte la même modification dans les deux conteneurs.

```yaml
version: '3'
services:
  nginx1:
    image: nginx:latest
    volumes:
      - nginx-volume:/usr/share/nginx/html
    ports:
      - "8081:80"

  nginx2:
    image: nginx:latest
    volumes:
      - nginx-volume:/usr/share/nginx/html
    ports:
      - "8082:80"

volumes:
  nginx-volume:
```

Chapitre 3 : Approfondissement Docker

Module 3.2 – Les volumes dans Docker Compose

Exemple de Docker compose

services :

nginx_service_1 et nginx_service_2: Deux services Nginx utilisant l'image Nginx officielle.

volumes : précise le volume qui est monté sur le conteneur (il est possible d'en préciser plusieurs)

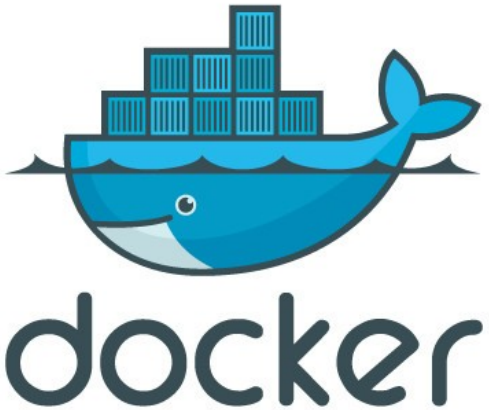
volumes : Définit les volumes nommés

nginx-volume : le nom du volume nommé

Chapitre 3 : Approfondissement Docker

Module 3.2 – Gestion des volumes et persistance des données

Manipulation des volumes (TP)



Chapitre 3 : Approfondissement Docker

Module 3.2 – Manipulation des volumes

Manipulation des volumes (TP)

Suivre le TP fournit par le formateur.

Chapitre 3 : Approfondissement Docker

Module 3.2 – Questions ?

Chapitre 3 : Approfondissement Docker

Module 3.3
Bonnes pratiques pour écrire des Dockerfiles

Chapitre 3 : Approfondissement Docker

Module 3.3 – Bonnes pratiques pour écrire des Dockerfiles

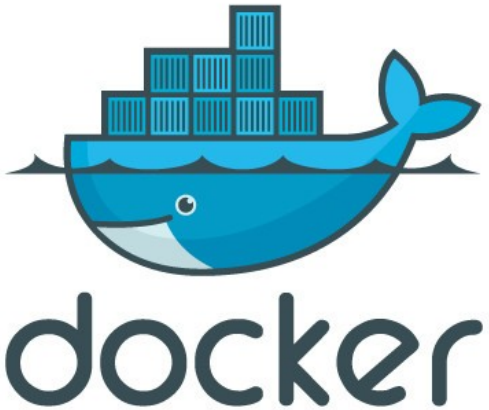
Sommaire :

- Les bonnes pratiques pour Dockerfile
- Écriture d’un Dockerfile avec les bonnes pratiques (TP)

Chapitre 3 : Approfondissement Docker

Module 3.3 – Bonnes pratiques pour écrire des Dockerfiles

Les bonnes pratiques pour Dockerfile



Chapitre 3 : Approfondissement Docker

Module 3.3 – Les bonnes pratiques pour Dockerfile

Rappel du fichier Dockerfile

Le fichier Dockerfile permet de décrire des instructions pour réaliser la construction d'une image Docker.

En respectant certaines bonnes pratiques, vous pouvez vous assurer d'obtenir une image :

- La plus légère possible
- réduire sa complexité
- améliorer sa maintenance
- respecter des notions de sécurité

Chapitre 3 : Approfondissement Docker

Module 3.3 – Les bonnes pratiques pour Dockerfile

Introduction aux bonnes pratiques

Le site https://docs.docker.com/develop/develop-images/instructions/ indique une liste de points pour écrire des Dockerfile en respectant les bonnes pratiques.

Dans ce module, nous allons aborder 5 bonnes pratiques :

- Utiliser la bonne image
- Minimiser le nombre de couches d'une image
- Ordonnancer les instructions d'un Dockerfile
- Gérer les labels
- Documenter votre Dockerfile

Chapitre 3 : Approfondissement Docker

Module 3.3 – Les bonnes pratiques pour Dockerfile

Bonne pratique 1 : Utiliser la bonne image de base

En fonction de votre besoin, utilisez une image adaptée à votre projet.

Par exemple, si vous réaliser une solution applicative qui utilise la technologie php, vous avez plusieurs solutions :

- Utiliser une image avec un OS spécifique (debian, centos, ubuntu, etc.) puis installer PHP par dessus
  - si vous avez besoin d'élément spécifique à l'OS on devra privilégié ce choix

- Utiliser une image de base avec php directement.
  - Si l'OS importe peu, alors privilégié ce choix, car il simplifiera votre image

De manière générale, utiliser plutôt les images officielles (car elles sont vérifiées) et les images en version « alpine » (distribution Linux ultra-légère) en raison de la taille contenue (inférieur à 6Mo actuellement).

Chapitre 3 : Approfondissement Docker

Module 3.3 – Les bonnes pratiques pour Dockerfile

Bonne pratique 2 : Minimiser le nombre de couches

Dans votre Dockerfile, chaque instruction va créer une nouvelle couche, afin de garder une image la plus lisible possible, réduire le nombre d'instruction va améliorer sa maintenance.

Les 2 exemples suivants vont effectuer le même résultat final, mais l'un est optimisé et l'autre non :

Exemple non optimisé :

```
RUN apt-get update
RUN apt-get install -y package-bar
RUN apt-get install -y package-baz
RUN apt-get install -y package-foo
RUN rm -rf /var/lib/apt/lists/*
```

Exemple optimisé :

```
RUN apt-get update && apt-get install -y \
    package-bar \
    package-baz \
    package-foo  \
    && rm -rf /var/lib/apt/lists/*
```

Dans le cas non optimisé, l'image aura 5 couches dans son historique, alors que dans l'exemple optimisé l'image aura une seule couche.

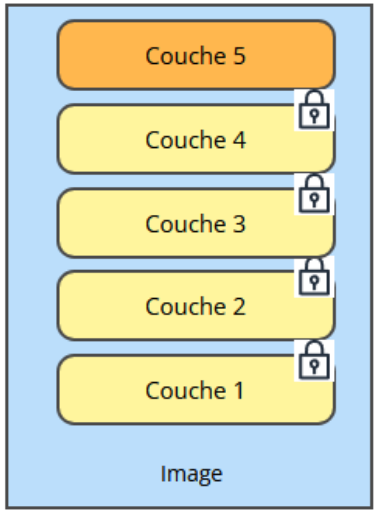
De plus, dans le cas d'un apt-get update et apt-get install, dans l'exemple non optimisé il peut y avoir des problèmes de cache.
En effet, le résultat d'apt-get update sera dans une couche de l'image, si dans la couche suivante on ajoute un nouveau paquet à installer, alors cela va poser un problème car il sera désynchroniser par rapport à la commande update précédente.

Chapitre 3 : Approfondissement Docker

Module 3.3 – Les bonnes pratiques pour Dockerfile

Bonne pratique 3 : Ordonnancer les instructions

Comme nous l'avons vu dans le chapitre précédent, une image Docker est réalisée sous forme de couches, il est donc préférable de placer en dernier les instructions qui sont les plus susceptibles de changer, de cette manière, on profitera toujours du cache et la base de l'image ne sera pas modifiée.



Chapitre 3 : Approfondissement Docker

Module 3.3 – Les bonnes pratiques pour Dockerfile

Bonne pratique 4 : gérer les Labels

Les Labels vont permettre d'identifier et de documenter votre image (auteur, version , entreprise, etc...).

```
# Set multiple labels at once, using line-continu
LABEL vendor=ACME\ Incorporated \
      com.example.is-beta= \
      com.example.is-production="" \
      com.example.version="0.0.1-beta" \
      com.example.release-date="2015-02-12"
```

Chapitre 3 : Approfondissement Docker

Module 3.3 – Les bonnes pratiques pour Dockerfile

Bonne pratique 5 : documenter votre Dockerfile

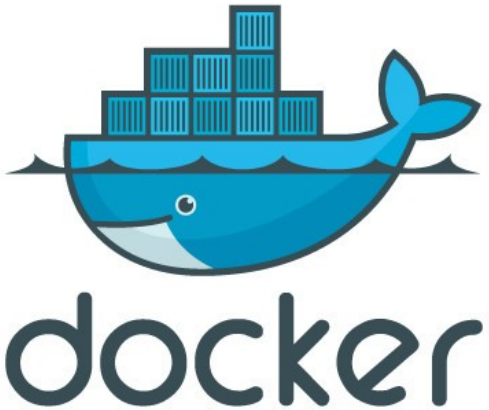
Lorsque vos instructions deviennent complexe, documenter votre Dockerfile pour faciliter la maintenance ultérieure.

```
# Étape 1 : Utiliser une image de base légère
FROM node:14-alpine

# Étape 2 : Définir le répertoire de travail dans le conteneur
WORKDIR /app

# Étape 3 : Copier les fichiers de configuration
COPY package.json .
COPY package-lock.json .

# Étape 4 : Installer les dépendances
RUN npm install

# Étape 5 : Copier le reste des fichiers de l'application
COPY . .

# Étape 6 : Exposer le port sur lequel l'application s'exécute
EXPOSE 3000

# Étape 7 : Définir la commande par défaut pour exécuter l'application
CMD ["npm", "start"]
```

Chapitre 3 : Approfondissement Docker

Module 3.3 – Bonnes pratiques pour écrire des Dockerfiles

Écriture d’un Dockerfile avec les bonnes pratiques (TP)



Chapitre 3 : Approfondissement Docker

Module 3.3 – Bonnes pratiques pour écrire des Dockerfiles

Écriture d’un Dockerfile avec les bonnes pratiques (TP)

Suivre le TP fournit par le formateur.

Chapitre 3 : Approfondissement Docker

Module 3.3 – Questions ?

Chapitre 3 : Approfondissement Docker

Module 3.4
Sécurisation des conteneurs

Chapitre 3 : Approfondissement Docker

Module 3.4 – Sécurisation des conteneurs

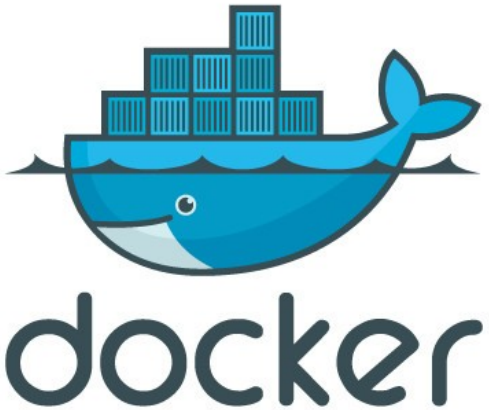
Sommaire :

- Introduction à la sécurité
- Gestion des images, des droits et des ressources
- Surveillance des images et conteneurs

Chapitre 3 : Approfondissement Docker

Module 3.4 – Sécurisation des conteneurs

Introduction à la sécurité



Chapitre 3 : Approfondissement Docker

Module 3.4 – Introduction à la sécurité

Introduction à la sécurité

La sécurité informatique est un domaine d’expertise qui vise à protéger les systèmes, les logiciels, les données et les réseaux contre des attaques malveillantes.

Les attaques malveillantes peuvent prendre plusieurs formes :

- Vol de données,
- Contrôle des systèmes,
- Usurpation d’identités,
- Dégât sur les appareils informatiques,
- Interruption de l’activité,
- Etc…

Chapitre 3 : Approfondissement Docker

Module 3.4 – Introduction à la sécurité

Investissement dans la sécurité

En fonction des moyens de l'entreprise, un budget adapté à sa structure doit être mobiliser pour la sécurité de son système d'information.

Généralement, ce budget représente 5 à 25 % du budget informatique total de l'entreprise.

Comment positionner le curseur de ce budget ?

Il faut au préalable établir les risques potentiels sur son business liés à une défaillance de sécurité.

Chapitre 3 : Approfondissement Docker

Module 3.4 – Introduction à la sécurité

Quelques chiffres

- 54 % des entreprises françaises ont été victimes d'attaques (source Baromètre Cesin 2022)
- 59 000€, le coût moyen d'une attaque (lorsque l'entreprise est victime d'une attaque) (source Etudes asteres)
- 69 % des sociétés ciblés sont des TPE/PME (source CNIL)
- 94 % des logiciels malveillants sont délivrées par mail (source Data Breach Investigations Report)

https://asteres.fr/site/wp-content/uploads/2023/06/ASTERES-CRIP-Cout-des-cyberattaques-reussies-16062023.pdf

https://www.arcsi.fr/doc/Barometre-cyber-entreprises-CESIN-Janvier-2022.pdf

Chapitre 3 : Approfondissement Docker

Module 3.4 – Introduction à la sécurité

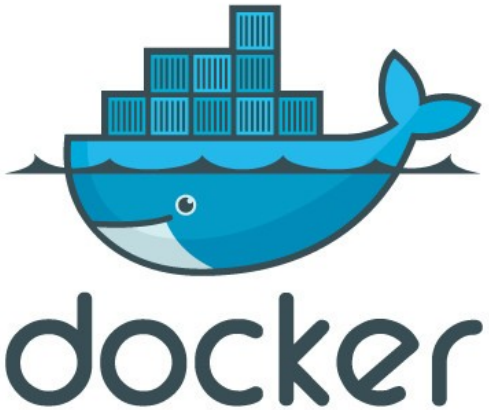
Conclusion

Il est important de prendre en compte l’aspect de la sécurité dans tous les composants d’un système d’information, c’est pourquoi nous devons aborder la question de la sécurité dans Docker.

Chapitre 3 : Approfondissement Docker

Module 3.4 – Sécurisation des conteneurs

Gestion des images, droits et ressources



Chapitre 3 : Approfondissement Docker

Module 3.4 – Gestion des images, droits et ressources

Gestion des images

Utiliser des sources fiables

En terme de sécurité, il est important d'utiliser les composants et librairies de fournisseurs vérifiés.
Pour Docker, dans le même principe au niveau des images, il faut privilégier les images officielles.

Utiliser uniquement ce dont vous avez besoin

Une bonne pratique concernant la sécurité est d'utiliser uniquement ce qui est nécessaire, ceci afin de réduire les composants possibles à attaquer.
Au niveau de Docker, cela se traduit par utiliser des images qui embarquent le strict nécessaire pour réaliser la tâche voulue.
Cela rejoint la bonne pratique de l'écriture du Dockerfile, à savoir de privilégié les images minimales (alpine, slim).

Chapitre 3 : Approfondissement Docker

Module 3.4 – Gestion des images, droits et ressources

Gestion des images

Reconstruire les images dans Docker

Une image Docker est générée par un Dockerfile qui contient un ensemble d'instructions.
Il faut se rappeler qu'une image est une vue instantanée lors de sa construction.

Pour se prévenir de certaines vulnérabilités, Docker recommande de mettre à jour ses images régulièrement en utilisant l'option –**no-cache** lors de la construction de l'image, de cette manière, Docker va récupérer les dernières versions téléchargeables des images et dépendances.

```
# syntax=docker/dockerfile:1
FROM ubuntu:latest
RUN apt-get -y update && apt-get install -y python
```

```
docker build --no-cache -t myImage:myTag myPath/
```

Chapitre 3 : Approfondissement Docker

Module 3.4 – Gestion des images, droits et ressources

Gestion des droits

Réduire les privilèges

Une mauvaise habitude est de lancer ses exécutables avec un compte root, car ce dernier dispose de privilèges qui permet d'accéder à des parties réservées du système et donc si cet utilisateur est compromis il pourrait faire des dégâts importants sur la machine.

Dans Docker, pour éviter l'utilisateur root, il est possible d'écrire l'instruction « USER » dans le Dockerfile pour spécifier un utilisateur avec moins de droits pour exécuter un service par exemple.

Chapitre 3 : Approfondissement Docker

Module 3.4 – Gestion des images, droits et ressources

Gestion des droits

Réduire les privilèges

Dans cet exemple, le Dockerfile utilise l'image officielle de Nginx basée sur Alpine Linux.

Un utilisateur non privilégié appelé **mynginxuser** est créé, et l'instruction **USER mynginxuser** spécifie que toutes les instructions suivantes seront exécutées en tant qu'utilisateur mynginxuser.

```
# Utilisation d'une image Nginx officielle basée sur Alpine
FROM nginx:latest

# Création d'un utilisateur non privilégié (non-root)
RUN adduser -D mynginxuser

# Définition de l'utilisateur par défaut pour les instructions suivante
USER mynginxuser

# Exposition du port 80 (Nginx)
EXPOSE 80

# Répertoire de travail pour les instructions suivantes
WORKDIR /usr/share/nginx/html

# Copie des fichiers de configuration personnalisés
COPY nginx.conf /etc/nginx/nginx.conf

# Copie des fichiers de l'application dans le répertoire web de Nginx
COPY . .

# Commande par défaut pour démarrer Nginx
CMD ["nginx", "-g", "daemon off;"]
```

Chapitre 3 : Approfondissement Docker

Module 3.4 – Gestion des images, droits et ressources

Gestion des droits

Le mode lecture seule

Vous pouvez verrouiller le système de fichiers du conteneur pour prévenir l'écriture, ensuite il faudra combiner avec la notion de volumes pour pouvoir déclarer au conteneur les chemins qui sont éditables :

```
docker run --read-only -v /icanwrite busybox touch /icanwrite/here
```

Chapitre 3 : Approfondissement Docker

Module 3.4 – Gestion des images, droits et ressources

Gestion des ressources

Définir les ressources du conteneur

Utilisez des contrôles de ressources tels que les limites de mémoire et de CPU pour empêcher les conteneurs de monopoliser les ressources du système.

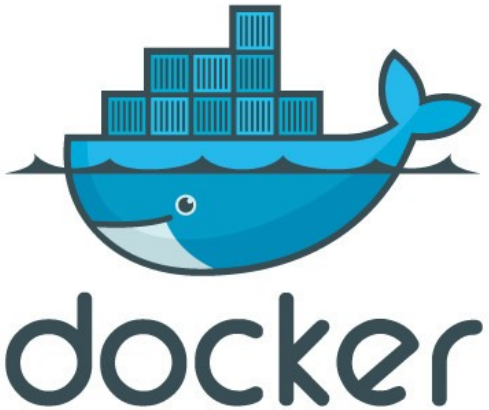
Cela aide à prévenir les attaques par déni de service.

```
docker run --memory=512m --cpu-shares=256 -dit --name ressources ubuntu bin/bash
```

Chapitre 3 : Approfondissement Docker

Module 3.4 – Sécurisation des conteneurs

Surveillance des images et conteneurs



Chapitre 3 : Approfondissement Docker

Module 3.4 – Surveillance des images et conteneurs

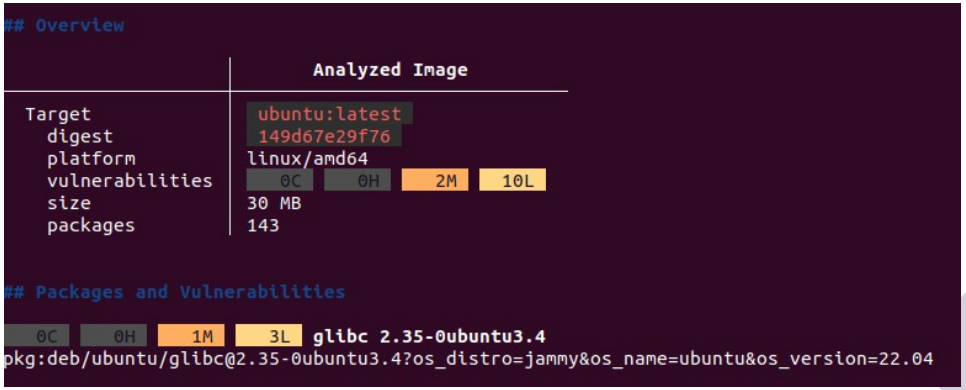
Vérifier les images pour détecter des vulnérabilités

Il existe des outils pour vérifier les vulnérabilités des images :

- Docker Hub permet la vérification automatique des images qui sont postés (nécessite un abonnement)
- Une commande **docker scout** permet de vérifier les vulnérabilités (L = Low, M= Medium, H = High, C = Critical)

```
docker scout quickview ubuntu
```


```
docker scout cves ubuntu
```

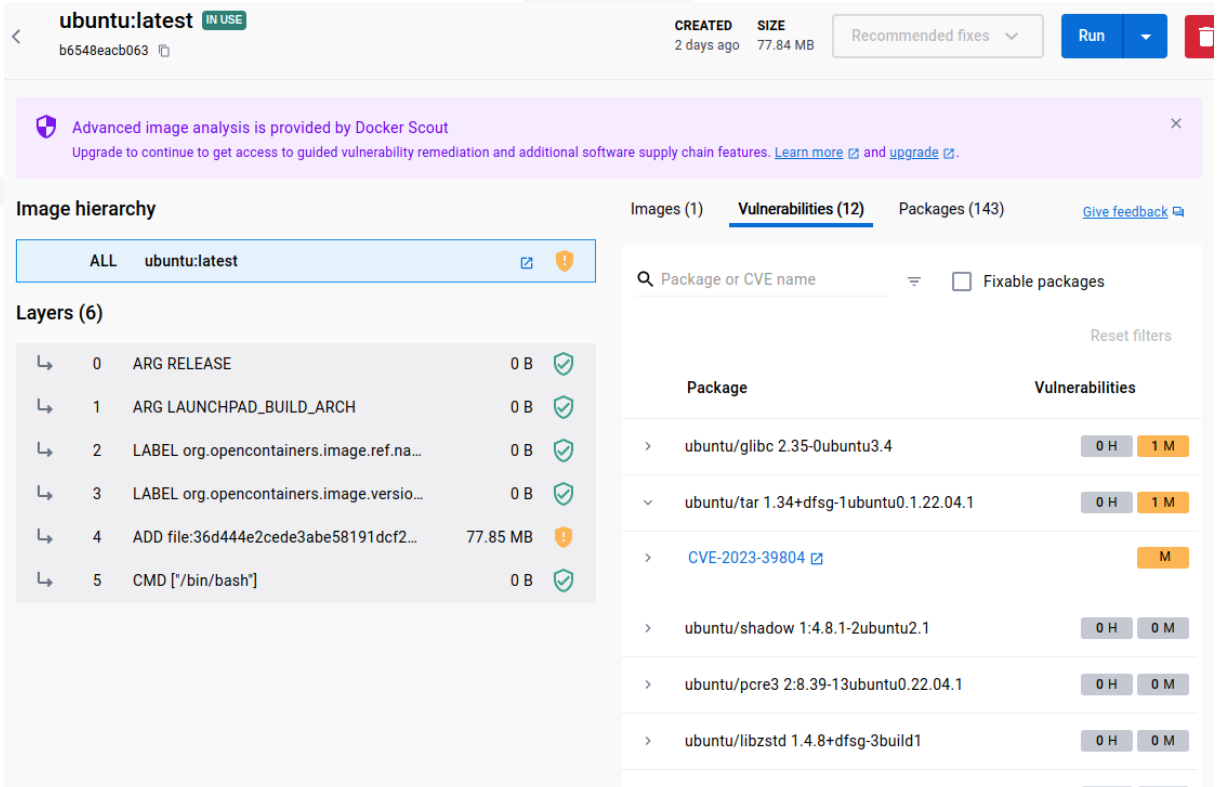


Chapitre 3 : Approfondissement Docker

Module 3.4 – Surveillance des images et conteneurs

Vérifier les images pour détecter des vulnérabilités

- Docker Desktop permet également de vérifier l'état de vulnérabilités des images



Chapitre 3 : Approfondissement Docker

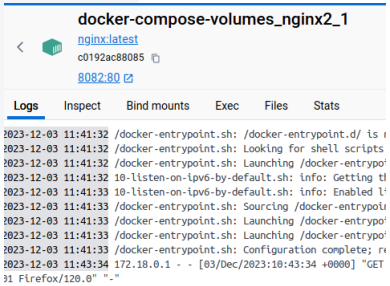
Module 3.4 – Surveillance des images et conteneurs

Surveiller les logs des conteneurs

Comme pour toutes applications, il faut surveiller les logs générés afin de détecter des comportements anormaux.

Les conteneurs produisent des logs accessibles sur Docker CLI et Docker Desktop :

- docker logs <conteneur>
- Sur Docker Desktop :



Pour faciliter cette démarche, il est possible d'utiliser une stack de solutions de gestion de logs centralisées ELK (Elastic search / Logstatsh / Kibana).

- Elastic Search : Stocke les données et permet la recherche à travers des APIs REST
- Kibana : Un plugin de Elastic Search qui permet la constitution de tablaux de bord
- Logstash : composant d'Elastic Search qui permet de collecter et traiter les données afin de les envoyer vers un service de stockage

Chapitre 3 : Approfondissement Docker

Module 3.4 – Questions ?