le guide de docker avec des schémas et des commandes interactives pour les gens largués qui ont besoin d'explications détailées et qui mangent du terminal

et les titres à rallonge aussi

lelu awen

Sommaire

- Avant-propos
- Les conteneurs en bref
  - À quoi que ça sert
  - Open Container Initiative (OCI)
- Docker
  - Présentation générale
  - Fonctionnement
    - Conteneurs
    - Images
    - Volumes
    - Ports
    - Networks
- Glossaire

Avant-propos

Le but de ce document est de fournir une base synthétique sur le sujet de l'utilisation de docker dans le cadre de la manipulation de conteneur de façon impérative. Ainsi, les sujets suivants ne serons pas aborder:

- Docker compose
  - un outil de permettant de définir un ensemble de conteneur (projet) de façon déclarative.
- Dockerfile
  - Mécanisme de docker permettant de créer des **Images**[1]

Ce Guide par du principe que vous avez **docker**[2] d'installer sur votre ordinateur afin d'exécuter les différentes commandes que ce dernier comprend
De plus, nous n'allons pas revenir en détail sur le fonctionnement d'un terminal, des diverses commandes non liées à docker et nous survolerons les services utilisés comme exemple.
De plus, ce guide vise à voir les fondamentaux de docker, de ce fait, il ne sera pas fait mention de **docker desktop** et de **lazydocker**, car bien que pratique, je juge qu'il est fondamental de bien connaître le fonctionnement interne de docker avant de se faire assister par des outils externes.

Pour toute question, merci de me contacter au mail suivant: lelu.awen@proton.me

Les conteneurs en bref

À quoi ça sert

Avez vous déjà eu le problème suivant:
Imaginez que vous êtes étudiant, vous avez un cours sur de l'intégration continue, à tout hasard, *jenkins*[3], Le Professeur vous demande donc d'installer Jenkins sur votre ordinateur et là, le drame, l'horreur, vous devez installer *java*[4]. Bien sur, vous allez voir sur la documentation d'installation de *Jenkins* et vous voyez que ce dernier demande la version 21 de Java, hors, vous avez la version 25 d'installer pour vos projets ! Que faire ? Passer vos projets sur java 21 ? Désinstaller java 25 et le réinstaller après le cours ? Dans tous les cas, c'est pas vraiment pratique. Maintenant, imaginez en plus que vous êtes sur Windows et que vos partenaires de groupe sont sur *Linux* et *MacOS*, ils ont chaqu'un des modalités d'installation différente.
Si seulement on pouvait avoir un système qui permet de faire en sorte que tout notre projet et ses dépendances soit toutes regrouper dans un paquet, et que vous ayez juste à télécharger ce paquet pour le lancer... Oh l'idée de génie vite faut que je lance ma startup avant qu'on me pique l'idée et on pourra même utiliser une IA pour.. Oh, c'est le sujet de la diapo, c'est ça ?

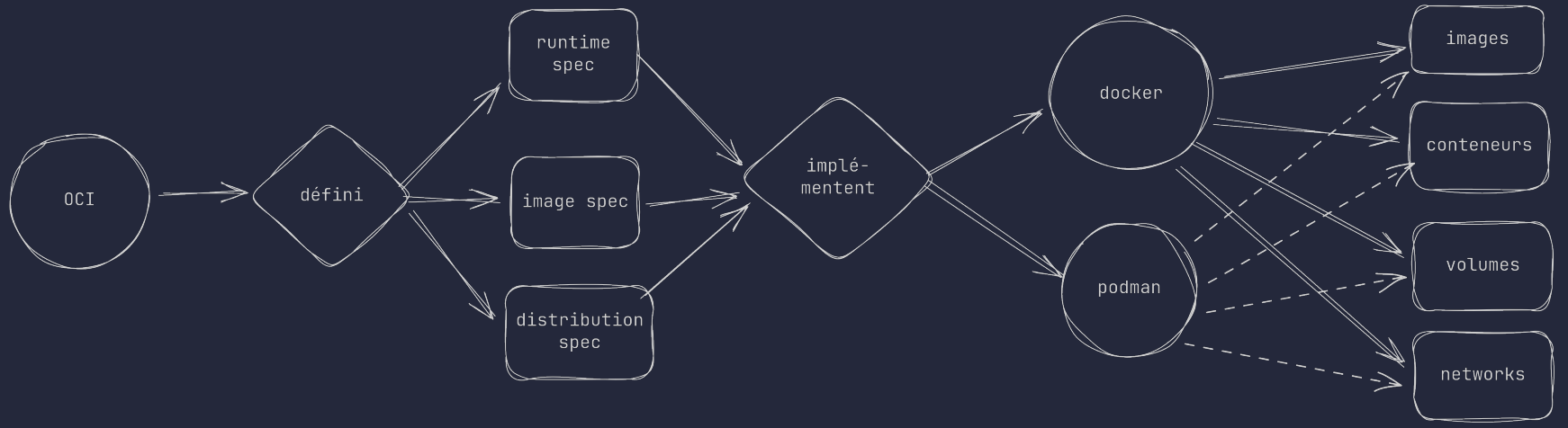
Et oui, un des buts des conteneurs c'est ça, pouvoir mettre ensemble tout les composants d'une application dans une boite, et pouvoir lancer cette boite de façon isolée du reste du système, un peu comme son propre petit ordinateur. de cette manière on gagne plusieurs avantages:

- ne pas avoir à gérer plusieurs versions des dépendences
  - comme par exempleflake_schema avoir 2 version de node d'installer, ou deux version de java
- ne pas risquer d'oublier des trucs qu'on à installer juste pour un projet
  - Car il est installer que au seins de notre conteneur, et si on supprime celui-ci, on supprime aussi les programmes associé.
- Pouvoir lancer des trucs "juste pour tester"
  - Chaque conteneur est isolé de notre système, donc il risque pas de casser des trucs, et vu qu'on peut le supprimer après, on risque pas d'oublier de le désinstaller
- Partager un environnement
  - quel que sois le système d'exploitation, maintenant tout le monde peut faire les même commande pour avoir strictement le même environnement de travail que le voisin, finis les "Ah mais ton truc il marche pas parce que j'ai java 11 d'installer"

Les conteneurs en bref

Open Container Initiative (OCI)

Populariser par Docker, l'utilisation de conteneur n'est pourtant pas une exclusivité de ce dernier. Ainsi il existe d'autres outils permettant d'utiliser des conteneurs, comme par exemple Podman. Afin de standardiser la définition de ce qu'est unn conteneur, Docker ainsi que d'autres acteurs ont mis en place **l'Open Containuer Initiative** ou OCI, une structure ouverte servant d'autorité pour définir les standards régissant les conteneurs dits "**OCI**[5]", un peu à la manière du *Système de mesure International* (SI) définissant par exemple le mètre, le gramme, ect...
Ces standards étant ouverts, n'importe qui peut les implémenter afin de construire son propre engine et pouvoir tout de même utiliser les systèmes existants.
Ainsi, bien que **Docker** et **Podman** sois deux implémentation différentes de ce standard, une image construite avec **Docker** peut parfaitement être utiliser avec **Podman** et vis-versa.
On parlera ainsi de **Conteneur OCI** pour parler d'un conteneur construit celons ce standard.



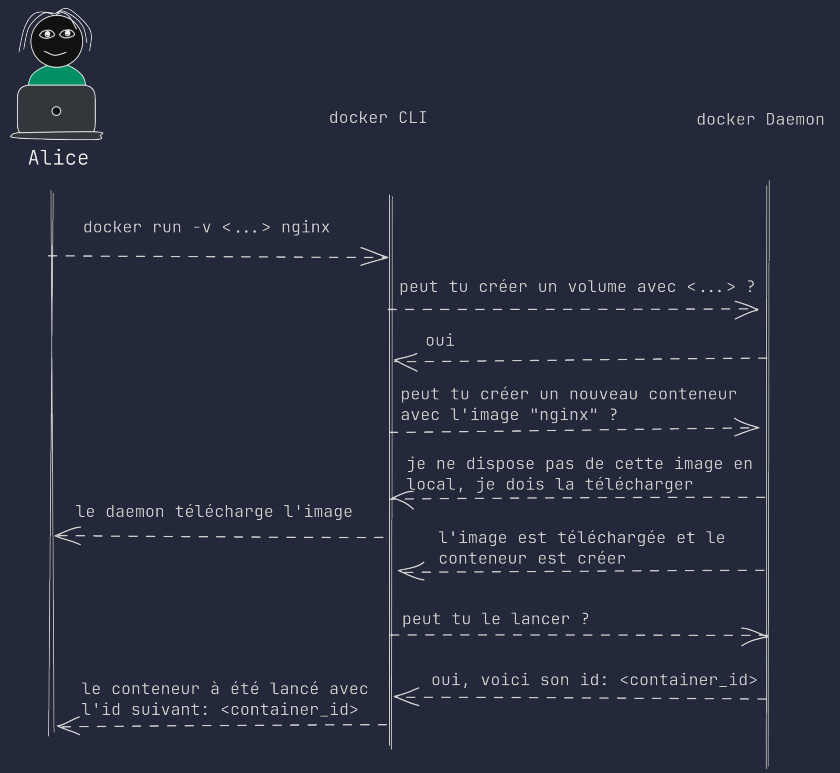
Docker

Présentation générale

Maintenant qu'on a passé la partie ennuyante, rentrons dans le vif du sujet:
Docker c'est quoi ?
Fondamentalement, Docker est un programme permettant de créer, lancer et gérer des **conteneurs**, des **volumes**, des **network** et des **images** respectant le standard OCI.
Il est formé de principalement deux composants :

- Docker daemon
  - Son rôle est de fonctionner en arrière plan de votre ordinateur et de chapeauté l'exécution des conteneurs, le routage des requêtes entre ces derniers via les *networks*[6] ainsi que l'accès aux fichiers de l'ordinateur via les *volumes*[7]
- Docker CLI
  - Ce composant fournis les commandes qui vont être utilisées pour manipuler le daemon, en pratique, lui demander de lancer un conteneur, créer une **image**, un **volume** ou un **network**.

En effet, pour rentrer vaguement dans les détails plus techniques, docker fonctionne avec une architecture client / serveur. Le daemon fonctionne en arrière-plan et est le "cœur" de Docker, tendis que les commandes servent juste à envoyer des requêtes à ce dernier.



Docker -- fonctionnement

Commandes

Ok après toute cette théorie, voyons les différentes commandes permettant d'interagir avec docker et plus particulièrement avec les images docker.
Tout d'abord, les commandes docker sont structurés en 4 parties:

```
docker <module> <action> <arguments>
```

En détail:

- **docker**: la commande de base pour appeller docker, toutes les commandes commencent par ça.
  - **<module>**: le nom du module de docker que l'on souhaite utiliser, chaque module sert à interagir avec une partie spécifique de docker. Par exemple, le module **image** sert à interagir avec les images docker.
  - **<action>**: L'action du module à réaliser, chaque module possède des actions qui sont lié à son utilisation. Par exemple, pour le module **image**, l'action **pull** sert à télécharger une image depuis dockerhub sur son ordinateur.
  - **<arguments>**: enfin, chaque action vas demander des arguments qui sont détailer dans le manuel de chaque action. par exemple, l'action **pull** du module **image** demander le nom de l'image à télécharger.

Dans la pratique, si je veut télécharger l'image **node:22**, la commande à composer sera la suivante:

```
docker image pull node:22
```

> Conseil
>
> Je vous invite à tester les commandes au fil qu'on les vois afin de les tester et de les mémoriser.

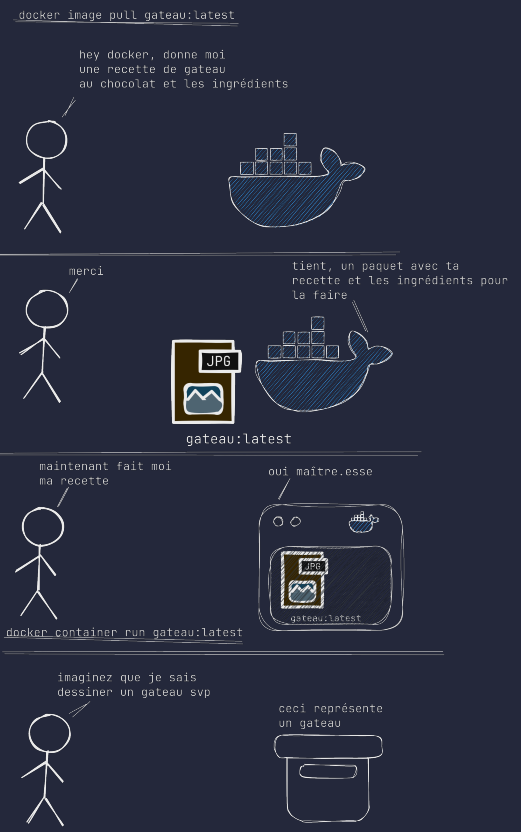
Docker -- fonctionnement

Images

Comme vu précédement, l'utilité de docker est de pouvoir créer et lancé des conteneurs isolé du système et qui agissent comme des mini ordinateurs. Mais pour cela, ils faut que le conteneur puisse savoir quoi lancé, de quels applications il à besoin pour fonctionner, et sur quel distribution[8] se baser.

C'est l'utilité de l'**image docker**[1], cette dernière sert en quelque sorte à définir les *ingrédients* dont notre conteneur vas avoir besoin pour se lancer correctement, contrairement au nom (et à mon schéma) peut laisser entendre, elle n'a rien d'une image au sens graphique du terme. En réalité, c'est plus quelque chose d'analog à un *fichier .zip* (ou .rar, ou .tar, ou .7zip, ou... bon vous avez l'idée...).

Cette image est composée de **layers**[9] (ou couches en bon francais), qui identifient et composent chaque élément de l'image. Cela permet à Docker de dédupliquer différents morceaux des images que l'on téléchargent.
Ainsi, si l'on télécharge deux images qui ont besoin de *nginx* par exemple, ces deux images vont partager un layer similaire, et ce layer n'aura besoin d'être télécharger qu'une seul fois.



nom des images

En règle générale, chaque image vas correspondre à un service, par exemple si veut lancer une *base de donnée mysql*, je vais utiliser une image nommée... *mysql*, pour un *serveur web nginx*, une image nommée... *nginx*, et si je veut un interpréteur python... *python*.

où trouver des images

Bon maintenant on sais à quoi ça sert, où est ce que je peut trouver quel nom d'image utiliser et si une image existe pour le service que je cherche ? Par exemple, mettons que je souhaite une image docker de *node* ?

La manière la plus simple est de chercher sur **dockerhub**[10] (*https://hub.docker.com/*) En reprenant notre exemple, si je cherche **node**, je tombe sur cette **page** (*https://hub.docker.com/_/node*), qui est une image officielle de docker, et donc subceptible d'être de bonne qualité, avec une documentation fiable et à jour.

Docker -- fonctionnement

Images

tag d'images

Maintenant, disons que l'on souhaite précisément la version 22 de node pour notre projet. On pourais chercher une image qui nous donne précisément cette version, mais généralement les images sont versionnée par ce qu'on appelle des *tags*, ils servent à spécifié une version précise d'une image, généralement construite avec des petites différences, que ce sois une version plus ancienne, ou une version spécifique.
Comment on spécificie en tag vous dites, tel le lecteur anthousiaste que vous êtes ? Eh bien la syntaxe est asser simple, on spécifie le nom de notr eimage comme d'habitude, suivis de deux points puis notre tag: `node:<tag>`. Mais ducoup, Comment savoir quoi mettre comme taf, redemendez vous d'une vois fébrile et avide de savoir, à moins que ce ne sois que les premiers symptômes de la schyzophrénie qui me frappent... C'est simple... Il suffit de regarder sur la page de documentation de **l'image en question** (*https://hub.docker.com/_/node*), dans les premières lignes, on trouvera généralement une liste des tags les plus courant, et en lisant plus en profondeur sur la documentation, on trouvera plus de détails sur l'utilisation de ces derniers. Ainsi, au châpitre **Image Variant** (*https://hub.docker.com/_/node#image-variants*), on trouve le paragraphe suivant:

> The node images come in many flavors, each designed for a specific use case.
>
> node:<version>
> This is the defacto image. If you are unsure about what your needs are, you probably want to use this one. It is designed to be used both as a throw away container (mount your source code and start the container to start your app), as well as the base to build other images off of.

On peut y lire (en anglais dans le texte), qu'on peut utiliser le la syntaxe `node:<version>` pour récupérer une version précise de node, ainsi, avec `node:22`, on vas récupérer une image avec la version 22 de node.

Docker -- fonctionnement

Images

Commandes utiles

```
docker image pull <nom_image>:<tag>
```

Cette commande permet de télécharger une image docker depuis internet sur votre ordinateur. ex: `docker image pull rancher/cowsay:latest`

> Tip
>
> Si aucun tag n'est préciser, la commande téléchargeta le tag *latest*

```
docker image list
```

Cette commande permet de lister les images présente sur votre ordinateur

```
docker image rm <nom_image>:<tag>
```

Cette commande permet de supprimer une image de votre ordinateur ex: `docker image rm rancher/cowsay`

> ⚠ Warning
>
> Vous ne pouvez pas supprimer une image en cours d'utilisation par un conteneur !
> Ainsi, vous devez d'abord supprimer les conteneurs utilisant cette dernière
> avant de la supprimer.

```
docker image search <recherche>
```

Cette commande permet de chercher des image docker correspondant à une requète, de la même façon que sur dockerhub, mais en ligne de commande. ex: `docker image search cowsay`

Docker -- fonctionnement

Conteneurs

```
docker run --rm rancher/cowsay:latest "test"
```

snippet +exec is disabled, run with -x to enable

Docker -- fonctionnement

- Volumes

Docker -- fonctionnement

Glossaire

1image docker:
2docker engine:
3jenkins:
4java:
5Conteneur OCI:
6docker networks:
7docker volumes:
8Distribution linux:
9layer d'image (docker):
10Dockerhub: