

**TP N°11 : Déploiement du service web Nginx dans un conteneur
géré à l'aide des composants logiciels de DOCKER****Durée : 1 séance de 2h00**

Moyens mis à disposition des étudiants	
Matériels : • 1 poste de travail sous Windows 10 ou 11 professionnel • Switch cisco SG 300-10	Logiciels et systèmes d'exploitation : • VirtualBox 7.2.6 et son pack d'extension • Debian Bookworm 12.11 • Docker Engine 29.3 • Docker Compose 5.1.1 • Contenaïrd 2.2.2 • Navigateur Internet (au choix)
Divers : Accès à Internet	

Savoir-faire mis en œuvre
❖ Justifier le choix d'une solution de mise en production d'un système ❖ Justifier le choix d'une solution de la disponibilité d'un serveur ❖ Installer et configurer une solution de disponibilité d'un serveur ❖ Sécuriser un serveur et une solution technique d'accès ❖ Administrer un système ❖ Valider et documenter une solution

Conditions de travail :

- Individuel
- **Le compte-rendu de ce TP sera rédigé sous la forme d'une documentation technique (page de garde + sommaire + bibliographie et/ou sitographie)** présentant les étapes d'installation, de configuration, de paramétrage ainsi qu'une phase de recette (les tests) des services déployés via Linux Debian Bookworm.
- L'infrastructure que vous devez mettre en place est présentée en annexes **document 1**.

Contexte professionnel :

La société les Jardins de Saint-Eloi est spécialisée dans la vente de fleurs exotiques et autres produits locaux (punch, rhum, sirop pays, ...) en Guadeloupe. Actuellement, la gérante Madame Bourgeois propose des compositions florales pour tous types d'évènements (anniversaire, mariage, baptême, enterrement, ...) et des idées cadeaux (constituée de compositions florales, punch, sirop pays, ...).

Ainsi, les clients (locaux et touristes) viennent directement dans l'entreprise pour passer commande et récupérer leurs compositions.

Madame Bourgeois a décidé de développer son activité dans toute la France. Pour cela, elle désire informatiser son activité en proposant aux clients de commander depuis une application web (site e-commerce) accessible via Internet.

Afin d'acheminer ses produits vers la France, Madame Bourgeois a conclu un partenariat avec le prestataire GPX Guadeloupe, société de transport international de marchandises. Ce dernier lui assure une livraison sous 48h ou 72h en fonction du pays et de la ville de livraison.

De plus, suite à la demande de certains clients, qui souhaitent réaliser seuls leurs compositions, Madame Bourgeois a décidé de mettre à leur disposition des fleurs, vendues au détail.

Madame Bourgeois a fait appel à l'ESN (Entreprise de services numériques) IPEOS I Solutions, implantée en Guadeloupe et en Martinique, afin de mener à bien ce projet.

Ce projet sera mené par un Développeur d'IPEOS I Solutions spécialiste dans la création d'applications métiers basées sur les technologies web traditionnelles et web 3.0.

Présentation des concepts et notions associés à la conteneurisation avec Docker

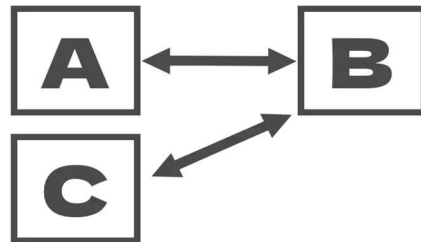
1. Présentation de Docker

Docker permet d'utiliser *plusieurs services applicatifs* grâce à des **conteneurs**. Ces conteneurs sont **indépendants les uns des autres, mais** qui **pourront communiquer ensemble, si nécessaire**.

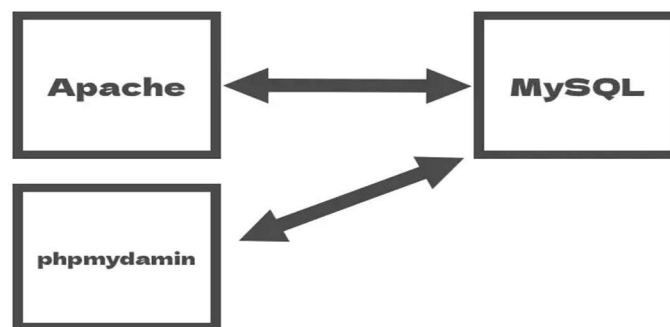
On peut assimiler un **conteneur** à une **enveloppe** ou une **boîte** qui **contient différents programmes utiles** au bon fonctionnement des services applicatifs.

Pour bien comprendre ce que signifient des conteneurs indépendants, mais qui peuvent être connectés, nous allons utiliser un schéma. On va nommer trois conteneurs A, B et C.

Dans cet exemple, le conteneur A et C ont besoin de communiquer avec le conteneur B pour bien fonctionner.



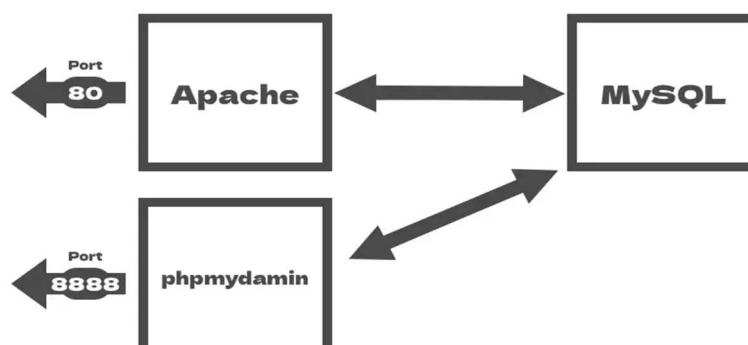
Bien entendu, **ceci est un modèle**, mais on peut utiliser et connecter autant de conteneurs que l'on souhaite. Utilisons ce **même exemple**, mais **avec des conteneurs qui existent** réellement et qui **sont disponibles**.



Sur ce schéma, nous utilisons **trois conteneurs** pour **trois services en interaction** :

- Il y a Apache qui a besoin du conteneur MySQL pour récupérer des données dans une base de données.
- Et nous avons un conteneur, phpmyadmin, qui doit également se connecter à MySQL pour récupérer des données.

Ensuite, **les services dont la fonction est de transmettre des informations** à la **machine hôte** et qui **doivent être lues ou modifiées** par un utilisateur, **le font grâce à un numéro de port**.



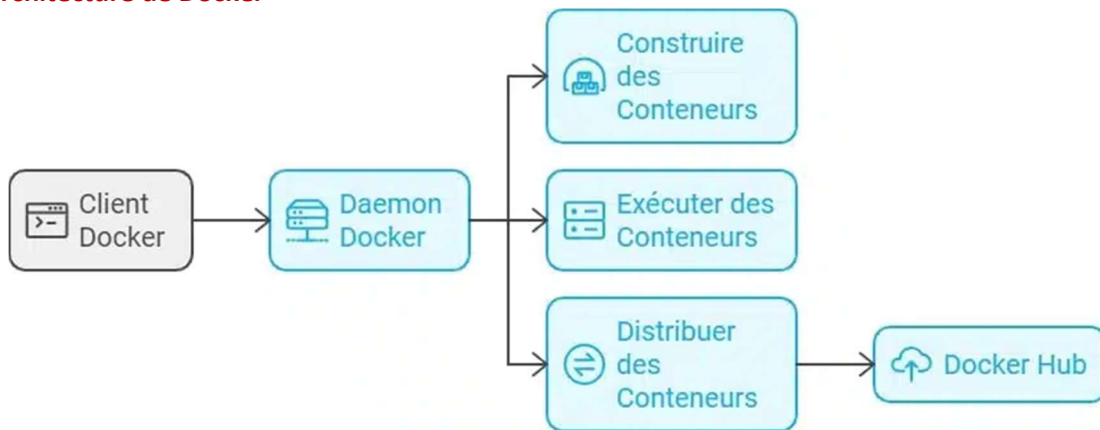
Dans notre cas, **l'utilisateur** de la machine hôte **pourra utiliser** le **port 80** pour **Apache** et le **port 8888** pour **Phpmyadmin**.

Observations :

- Une des forces de Docker, est la possibilité de démarrer et d'arrêter des services très rapidement à partir de **modèles enregistrés** appelés « **images** ».

- Il est donc tout à fait possible de créer une certaine configuration (avec plusieurs conteneurs) et de l'enregistrer dans un fichier texte, afin de les déployer dans d'autres machines très simplement.

2. L'architecture de Docker



Docker **repose sur une architecture client-serveur**. Le **client Docker** communique avec le **daemon Docker** qui se charge de construire, exécuter et distribuer les conteneurs. Les **registres** comme **Docker Hub** sont des dépôts qui stockent les images Docker. Ils permettent de partager et distribuer facilement ces images prêtes à l'emploi.

3. Le vocabulaire de Docker

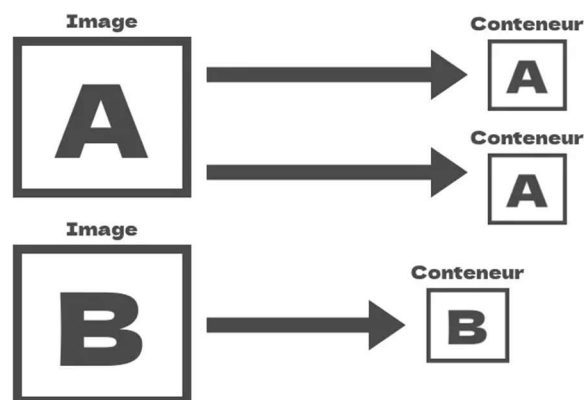
Pour bien comprendre et assimiler le fonctionnement de Docker, il faut dans un premier temps définir plusieurs expressions qui seront utilisées tout au long du TP :

- **Conteneur** : On a déjà évoqué le concept de conteneur précédemment. C'est comme une enveloppe qui contient différents programmes pour le bon fonctionnement d'un service « logiciel ».

Mais comment peut-on créer un conteneur ? Grâce à une image.

- **Image** : contient toutes les informations nécessaires pour créer un conteneur. Ou en d'autres termes, le conteneur va se baser sur une image pour se construire.

Remarque : Pour les développeurs et qui connaissent la programmation orientée objet, l'image est comme une classe et le conteneur est une instance de cette image.



Dans cet exemple, nous avons la représentation de deux images (A et B). Ces **deux images sont différentes grâce à leurs formes**.

- Ces **formes** symbolisent la **quantité** et le **contenu des programmes** que **chaque image contient**. Comme vous pouvez le voir, nous avons créé deux conteneurs à partir de l'image A et un conteneur à partir de l'image B.

Qu'est-ce que cela signifie ? Les deux conteneurs A contiennent les informations ou programmes que contiennent l'image A.

Par contre, les deux conteneurs sont indépendants l'un de l'autre. Si nous modifions un des conteneurs, la modification n'est pas reproduite dans l'autre.

Bien évidemment, elle n'est pas reproduite dans l'image qui est un modèle copié lors de la création du conteneur. De même, le conteneur B prend pour base l'image B.

- **Docker Hub** : est une **archive** (un **dépôt**) contenant toutes les **images disponibles (officielles ou non officielles)** sur le site officiel de l'éditeur Docker. Toutes ces images sont disponibles et chaque utilisateur a la possibilité de les télécharger grâce à une commande (nous verrons plus tard comment faire).

Remarque : Les images officielles sont révisées et publiées par Docker.

- **Dockerfile** : est un fichier dont le nom doit être absolument "**Dockerfile**" *et ne doit pas contenir d'extension*. **Ce fichier contient des informations qui nous permettront de créer une image à partir d'une autre image** et ainsi **obtenir une « image personnalisée »**. En effet, une image est basée sur une autre.

Par exemple, dans le Docker hub, il existe une image toute simple d'Ubuntu. Grâce à un Dockerfile, on pourrait créer une autre image qui se baserait sur Ubuntu, mais en plus, on ajouterait un programme (par exemple PHP). Nous aurons ainsi une nouvelle image avec Ubuntu et PHP.

Remarque : Grâce à cela, lorsque nous créerons un conteneur basé sur notre image personnalisée (Ubuntu + PHP), celui-ci contiendra directement Ubuntu et PHP.

- **docker-compose.yml** : est un fichier qui permet d'utiliser plusieurs conteneurs et de les connecter si nécessaire. C'est un outil puissant qui donnera la possibilité de rendre certains fichiers, choisis par nous-mêmes persistants. Nous verrons ensemble plus tard ce que cela signifie.

Remarque : L'extension **.yaml** (Yet Another Markup Language - YAML signifie n'est pas un langage de balisage) mais une façon de **structurer des données**, selon un format, et de les échanger. C'est un équivalent de .xml ou .json (données textuelles) mais avec un format différent.

- **GPG** : est un système de chiffrement qui permet de protéger vos données à l'aide d'un système de paires de clés publique-privée qui respectent la spécification OpenPGP.

Pour mieux appréhender les composants logiciels de Docker, différents documents sont mis à votre disposition en annexes :

Document 1 : Architecture réseau à réaliser

Document 2 : L'architecture client-serveur de Docker

Document 3 : Les commandes essentielles pour manipuler des conteneurs sous Docker

Document 4 : Méthodes de création de conteneurs et la plateforme d'images Docker Hub

Document 5 : Représentation du réseau interne de Docker Engine

Document 6 : Les 6 types (drivers) de réseaux Docker

Document 7 : Trois types de montage de volumes et persistance des données sous Docker

Travail à faire :

Etape 1 : Création de la VM « SRVDOCKER » et installation des composants logiciels de Docker

1. Créez ou clonez une machine virtuelle sous Linux Debian nommée « *srvdocker* » (*sans interface graphique*), selon la configuration suivante :

Nom	srvdocker
Disque dur	20 Go contrôleur SATA
Mémoire	4 Go
CD/DVD	Contrôleur IDE
Carte réseau	Active

2. Il y a deux possibilités pour **installer les composants logiciels de Docker**. Nous le ferons **depuis les serveurs de dépôts de Debian Bookworm** ainsi que **le dépôt officiel de Docker** à l'aide de l'utilitaire « *apt* » :
 - Mettez à jour vos sites miroirs (dépôts des paquets Debian) dans le fichier « *sources.list* ».
 - Saisissez les commandes suivantes pour effectuer la mise à jour de votre système Debian : **#apt update && apt upgrade**
 - Installez les paquets suivants : **ca-certificates, curl, gnupg et lsb-release**
3. Créez un répertoire pour les clés de sécurité pour télécharger de façon sécurisée les composants logiciels nécessaires avec la commande suivante :
 - **#sudo install -m 0755 -d /etc/apt/keyrings**
 - Expliquez le rôle de la commande ainsi que celui des options associées.
4. Téléchargez et ajoutez la clé GPG officielle de Docker :
 - **#curl -fsSL https://download.docker.com/linux/debian/gpg | gpg --dearmor -o /etc/apt/keyrings/docker.gpg**
 - Expliquez le rôle de la commande ainsi que celui des options associées.
5. Modifiez les droits d'accès du fichier « **docker.gpg** » en **ajoutant le droit en lecture à tout le monde**. Indiquez la commande que vous avez saisie.
6. A l'aide de la commande suivante, créez un fichier nommé « **docker.list** » à l'emplacement suivant « **/etc/apt/sources.list.d/** » :
 - **#echo « deb [arch=\$(dpkg --get-architecture) signed-by=/etc/apt/keyrings/docker.gpg] https://download.docker.com/linux/debian \$(. /etc/os-release && echo « \$VERSION_CODENAME ») stable » | sudo tee /etc/apt/sources.list.d/docker.list**
7. Editez le fichier « **docker.list** ». Que remarquez-vous ? Expliquez ce résultat en déclinant le rôle des commandes et option associées.
8. Actualisez vos fichiers sources à l'aide de la commande : **#apt update**
9. Maintenant, installez Docker CE et ses composants logiciels depuis la commande :
 - **#apt install docker-ce docker-ce-cli containerd.io docker-buildx-plugin docker-compose-plugin**
10. Vérifiez l'état du service « **Docker Engine** » :
 - **#systemctl status docker**
11. Configurez le démarrage automatique de Docker au lancement de « **SRVDOCKER** » :
 - **#systemctl enable docker.service**
 - **#systemctl enable containerd.service**

12. Arrêtez, démarrez et redémarrez le service Docker Engine à l'aide de la commande suivante :
 - **#systemctl start|stop|restart docker**
13. Affichez les informations relatives à la version de Docker installée sur votre serveur « **SRVDOCKER** », à l'aide des commandes suivantes :
 - **#docker version** ou **-v #docker info** **#docker compose version**

Remarque : En cas de message d'erreur, relancez votre serveur « **SRVDOCKER** ».

14. Testez chacune des commandes suivantes : **\$docker ps** et **\$docker ps -a**. Indiquez leur rôle ainsi que les options associées. Que constatez-vous ?

Etape 2 : Configuration de Docker et test du conteneur de démonstration « HelloWorld »

Vous allez configurer le compte utilisateur « **tepson** » afin qu'il ait accès aux services de Docker en mode « **sans élévation de privilèges** » encore appelé en mode « **moindre privilèges** » :

15. Pour ce faire, ajoutez l'utilisateur « **tepson** » au groupe utilisateur « **docker** » déjà existant à l'aide de la commande « **usermod** ». Indiquez la commande que vous avez saisie.
16. Quittez le mode administrateur et depuis le compte de l'utilisateur « **tepson** », lancez le conteneur de démonstration « **HelloWorld** » à partir de la commande : **\$docker run hello-world**

Remarque : En cas de message d'erreur, relancez votre serveur « **SRVDOCKER** ».

17. Testez et indiquez le rôle de chacune des commandes suivantes : **\$docker ps** et **\$docker ps -a**.

Poursuivons la configuration afin de personnaliser « **Docker Daemon** » lors de prise en charge de l'exécution des conteneurs :

18. Depuis le compte « **root** », créez un fichier JSON nommé « **daemon.json** » à l'emplacement « **/etc/docker** » pour personnaliser la configuration de « **Docker** » comme suit :

```
{
  «log-driver»: «json-file»,
  «log-opts» : {
    «max-size»: «10m»,
    «max-file»: «3»
  },
  «data-root» : «/var/lib/docker»,
  «live-restore»: true
}
```

19. Après avoir présenté le rôle et la structure d'un fichier JSON, expliquez l'intérêt du fichier « **daemon.json** » dans un contexte de « **conteneurisation** ».
20. Présentez le rôle de chaque paramètre du fichier « **daemon.json** ».
21. Maintenant, redémarrez « **Docker** ». Indiquez la commande saisie.

Les configurations réseaux physiques et logiques des conteneurs sur Docker sont très spécifiques (**voir documents 5 et 6 en annexes**).

Avant d'aller plus loin, vous devez comprendre comment « **Docker** » gère ses différents types (drivers) de réseaux.

22. A l'aide de la commande « **\$ip a** », listez les interfaces réseaux disponibles sur votre serveur Linux Debian « **SRVDOCKER** ». Comparez vos résultats avec le **document 5** présenté **en annexes**. Que concluez-vous ?

Etape 3 : Gestion des images et des conteneurs sur Docker

D'abord, il faut faire la différence entre une « **image** » et un « **conteneur** ».

Une **image Docker** est un package (ensemble de fichiers exécutables ou non, de répertoires liés à une application ou un logiciel et à son bon fonctionnement) en accès en lecture seule contenant tout le nécessaire pour exécuter cette application ou logiciel :

- Système de fichiers de base (OS minimal)
- Dépendances et bibliothèques
- Code de l'application
- Variables d'environnement
- Commande de démarrage

« **Docker** » met à la disposition des professionnels (développeurs et administrateurs) une plateforme en ligne, « **Docker Hub** » depuis laquelle ils pourront télécharger des images existantes (**officielles, sécurisées ou non**) mais aussi téléverser **leurs propres images**, qu'ils auront **personnalisés**.

Remarques :

- Il est fortement recommandé de créer ses conteneurs à partir d'images officielles, c'est-à-dire vérifiées et validées par les éditeurs de « **Docker** ».
- Des fichiers « **images** » de Docker sont déjà disponibles sur votre serveur « **SRVDOCKER** » et sont stockées dans le répertoire « **/var/lib/docker** ».

23. Listez les images stockées en locale (ou encore images locales) à partir des commandes suivantes : « **\$docker images** ou **\$docker image ls** ». Que constatez-vous ?

24. Indiquez le résultat obtenu ainsi que les informations fournies sur les images locales.

Maintenant, nous allons voir comment rechercher, télécharger et installer une image officielle de Docker.

25. Recherchez les images basées sur le système Linux Ubuntu à l'aide de la commande : « **\$docker search ubuntu** ». Indiquez les résultats obtenus ainsi que les informations fournies.

26. Téléchargez la dernière version du système « **Ubuntu** » à l'aide de la commande : « **\$docker pull ubuntu:latest** ». Saisissez la commande « **\$docker images** ». Que constatez-vous ?

Un **conteneur** est une **instance d'une image en cours d'exécution**. C'est un **processus isolé** avec son propre :

- Système de fichiers (copie de l'image + layer/couche d'écriture)
- Espace réseau
- Arborescence de processus (PID)
- Nom d'hôte (hostname)

27. Vérifiez qu'aucun conteneur autre que celui de « **HelloWorld** » est présent sur « **SRVDOCKER** » à l'aide des commandes : « **\$docker ps** puis **\$docker ps -a** ».

Remarques :

- Il existe de nombreuses commandes « **Docker** » pour créer, supprimer, lancer, arrêter, mettre en pause, reprendre, ... en résumé, pour gérer les conteneurs basés sous Docker. En conséquence, la suite du TP a vocation à aborder les principales commandes.
- Notez que pour créer un conteneur à partir d'une image Docker, il faudra d'abord utiliser la commande : « **docker run ...** ». Sachant que « **docker run = docker create + docker start** ».

28. Créez un conteneur nommé « **srvtest** » à partir de l'image « **Ubuntu** » préalablement téléchargée à partir de la commande : « **\$docker run -tid -name srvtest ubuntu:latest** ». Précisez le rôle de chaque paramètre et option associées à cette commande.

29. Vérifiez que votre conteneur est bien en cours d'exécution en saisissant : « **\$docker ps** » puis **\$docker ps -a** ». Indiquez et commentez les résultats obtenus en justifiant les informations fournies.
30. Accédez au terminal du conteneur « **srvtest** » à partir de la commande : « **\$docker exec -it srvtest bash** ». Que constatez-vous ? Testez les commandes suivantes : « **#hostname** » puis **#cat /etc/os-release** ». Commentez et justifiez les résultats obtenus.
- Remarque :** la commande « **#exit** » permet de quitter le terminal d'un conteneur et reprendre la main/revenir sur la machine hôte « **SRVDOCKER** ».
31. Proposez des commandes Docker pour arrêter l'exécution du conteneur « **srvtest** » et vérifiez que ce dernier est bien arrêté.

C'est à vous de jouer !!!

32. Indiquez les commandes à saisir pour : créez un conteneur nommé « **srvprod** » basé sur l'image de la dernière version d'Ubuntu, vérifier son état, accéder à son terminal, indiquer le nom du système et la version de Ubuntu installée.
33. Maintenant, indiquez la commande pour relancer le conteneur « **srvtest** » et précisez la commande permettant de voir, d'afficher tous les conteneurs en cours d'exécution.

Remarque : A ce stade, « **srvtest** » et « **srvprod** » sont deux instances en cours d'exécution de l'image **Docker Ubuntu latest**.

34. A l'aide de la commande « **\$docker rmi NOM_IMAGE** », supprimer l'image Docker de la dernière version de Ubuntu. Que constatez-vous ? Que faudra-t-il donc faire pour réaliser cette opération ? Justifiez !
35. Finalement, proposez une procédure pour supprimer les **images de Ubuntu** et les **conteneurs** basés dessus, à savoir « **srvtest** » et « **srvprod** ».

Observations : Jusqu'à présent, nous avons déployé des conteneurs basés sur une image de la dernière version de Ubuntu.

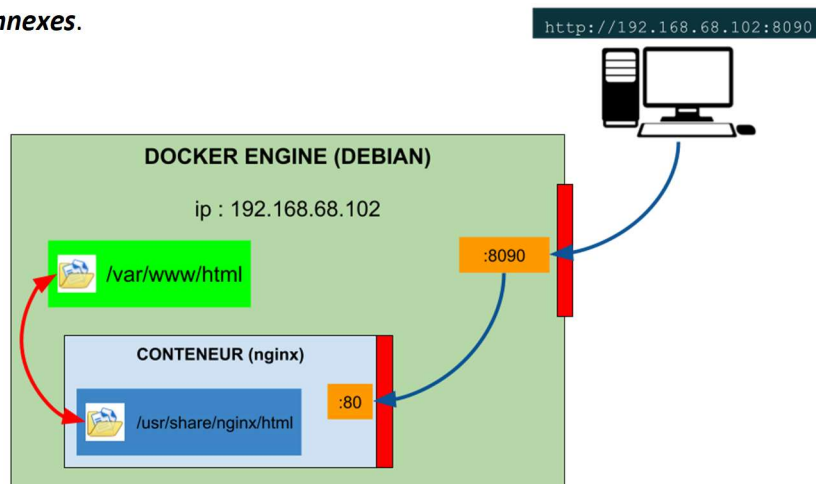
Or, même si c'est un OS léger et adapté pour la conteneurisation, nous embarquons tout de même un OS. Ce qui selon moi va à l'encontre des objectifs/de la finalité d'une solution de conteneurisation, à savoir déployer dans les conteneurs uniquement les composants, bibliothèques, dépendances logiciels utiles au bon fonctionnement d'une application. Pour conclure, un ou des conteneur.s sans OS embarqué.

Etape 4 : Exposition d'un conteneur « Nginx » au travers du réseau Docker

Dans la suite du TP, vous allez déployer le **service web Nginx** au sein d'un conteneur nommé « **srvweb** ».

Vous verrez également comment exposer (ou encore publier) un conteneur sur le réseau local de l'hôte ainsi que les ports de communication (TCP ou UDP).

Pour ce faire Docker Engine intègre un système de routage NAT/PAT sur un réseau qualifié de « **Bridge – Pont** », voir **document 5 et 6 en annexes**.



NGINX, c'est un logiciel très populaire qui sert principalement de **serveur web**. **NGINX** est un **outil clé du web moderne** : il permet d'héberger des sites, sécuriser les connexions et gérer le trafic efficacement.

Quelques consignes à suivre avant de débiter :

- Le **port du conteneur** « **srvweb** » sur lequel **écouter le service web Nginx** sera le **numéro 80**.
- Le **port d'exposition du conteneur** « **srvweb** » depuis **le réseau local** sera le **numéro 8080**.
- Un **poste de travail** appartenant au réseau logique servira de **poste « client »** pour **tester l'accès au conteneur et au service web Nginx hébergé**. Le poste « client » peut être votre ordinateur physique (portable ou bureau).
- Le fichier par défaut « **index.html** », c'est-à-dire la **page d'accueil de Nginx** est enregistré à l'emplacement « **/usr/share/nginx/html** » dans le conteneur « **srvweb** ».

36. Indiquez la commande Docker permettant de rechercher les images contenant les composants logiciels de « **Nginx** ».
37. Indiquez la commande Docker permettant de télécharger une image contenant uniquement « **Nginx** ».
38. Indiquez la commande Docker permettant de vérifier la présence effective de « **Nginx** » dans la liste des images Docker disponibles sur votre serveur « **SRVDOCKER** ».
39. Complétez la commande Docker suivante afin de créer et lancer le conteneur « **srvweb** » en respectant les consignes : « **\$ docker run -d --name NOM_CONTENEUR -p PORT_HOTE :PORT_CONTENEUR NOM_IMAGE** ». Précisez le rôle des options associées.
40. Indiquez la commande permettant de vérifier que le conteneur « **srvweb** » est bien en cours d'exécution. Précisez les informations relatives au conteneur « **srvweb** » qui sont fournies.
41. A l'aide la commande suivante, observez le contenu du fichier de configuration du conteneur « **srvweb** » : « **\$docker inspect srvweb | more** ». Tapez sur la touche « **Entrée** » pour faire défiler. Le raccourci clavier « **Ctrl + C** » + touche « **Entrée** » permettent de quitter ce fichier.
42. A l'aide de la commande suivante, affichez les informations des configurations IP de Docker Engine ainsi que du conteneur « **srvweb** » : « **\$docker network inspect bridge | more** ».
43. Précisez les informations sur le conteneur « **srvweb** » ci-après : adresses IP réseau, machine et MAC, masque de sous-réseau, passerelle, version de Nginx et status/état du conteneur. Comparez ces informations du réseau avec celles fournies en **annexes document 5**. Que constatez-vous ?
44. Enfin, indiquez la commande Docker permettant d'accéder au conteneur « **srvweb** » en ligne de commande (via le terminal bash) afin de modifier la page d'accueil par défaut de Nginx « **index.html** » stocké dans le répertoire « **/usr/share/nginx/index.html** » :
 - Il faut pouvoir accéder au terminal en mode interactif.
 - Proposez une procédure pas à pas pour modifier cette page « **index.html** » dont le nouveau contenu sera « **Bienvenue sur le site des Jardins de Saint-Eloi !** » ainsi que son titre qui sera « **Les Jardins de Saint-Eloi** ».

Remarque : l'utilitaire « **APT** » est disponible sur le conteneur « **srvweb** » pour mettre à jour et à niveau votre conteneur mais surtout pour installer les paquets dont vous jugerez nécessaires pour aboutir dans votre procédure.

45. Proposez une procédure pour tester l'accès au site web « **Les Jardins de Saint-Eloi** » hébergé dans le conteneur depuis un poste « **client** » de votre choix. Sachant que le conteneur est accessible depuis l'adresse IP et port exposés depuis Docker Engine.

Remarque : Le problème avec les conteneurs c'est que s'ils sont relancés depuis leur image, les modifications apportées à un fichier tel que « **index.html** » seront perdues.

46. Réutilisez la commande trouvée à la **question n°44**. Maintenant, essayez d'accéder au site web « **Les Jardins de Saint-Eloi** ». Que constatez-vous ?

Etape 5 : La gestion des volumes et la persistance des données.

Pour pallier au problème nous allons voir maintenant comment on peut mettre en place cette persistance (conservation) des données.

Observations : Les **conteneurs** sont **éphémères par nature** : quand un conteneur est supprimé, ses données disparaissent avec lui. Les **volumes** Docker résolvent ce problème en offrant un mécanisme de **persistance des données** indépendant du cycle de vie des conteneurs.

Trois options de montage des volumes s'offrent à vous : **volumes** standards (gérés par Docker), **bind mounts** (accès direct au système de fichiers hôte) et **tmpfs** (stockage en RAM), voir **document 7 en annexes**.

Il conviendra de bien **distinguer** le « **stockage local** » et « **stockage réseau** » sous Docker, voir **document 7 en annexes**.

Pour garantir la persistance des données, vous allez créer des volumes standards entre la machine hôte « **SRVDOCKER** » et le conteneur « **srvweb** ».

Docker propose **deux syntaxes** pour monter des volumes :

- La syntaxe **-v** est plus courte, plus concise, suffisante pour les cas simples.
- La syntaxe **--mount** est plus explicite et plus lisible et recommandée. Elle échoue avec une erreur claire si le chemin source n'existe pas (bind mount), indispensable pour les options avancées.

47. Indiquez les commandes Docker pour arrêter et supprimer le conteneur « **srvweb** ».

48. A partir du schéma présenté à l'étape 4, complétez la commande Docker suivante pour créer un conteneur « **srvweb** » basé sur Nginx, créer un volume représenté par le répertoire « **/var/www/html** » sur la machine hôte « **SRVDOCKER** » afin de garantir la persistance des données du site web « **Les Jardins de Saint-Eloi** » dont les pages html sont stockées à l'emplacement « **/usr/share/nginx/html** ». Les **ports d'écoute** côté **machine hôte** sera « **8080** » et côté **conteneur**, « **80** » :

- « **\$docker run -d --name NOM_CONTENEUR -p PORT_HOTE:PORT_CONTENEUR -v chemin_volume_hote:chemin_donnee_app NOM_IMAGE** »

Remarques :

- Pour la persistance des données Docker va ainsi monter (et créer au besoin) le répertoire « **/var/www/html** » de la machine hôte (Docker Engine) à l'emplacement « **/usr/share/nginx/html** » dans le conteneur. Si les répertoires n'existent pas, ils seront créés (et le propriétaire sera « **root** »).
- Il suffit donc maintenant de déposer (ou créer) dans « **/var/www/html** » de l'hôte un fichier « **index.html** » personnalisé afin de le distinguer de la page par défaut de Nginx et de vérifier que c'est bien la page d'accueil du site des Jardins de Saint-Eloi qui est publiée.

49. Créez le fichier « **index.html** » dans le volume créé sur la machine hôte « **SRVDOCKER** », « **/var/www/html** », et ajoutez-y le code html pour afficher la page d'accueil du site des Jardins de Saint-Eloi.

50. Testez l'accès au site web « **Les Jardins de Saint-Eloi** » hébergé dans le conteneur depuis un poste « **client** » de votre choix. Que concluez-vous ?

51. A l'aide des commandes Docker, arrêtez et supprimez le conteneur « **srvweb** ».

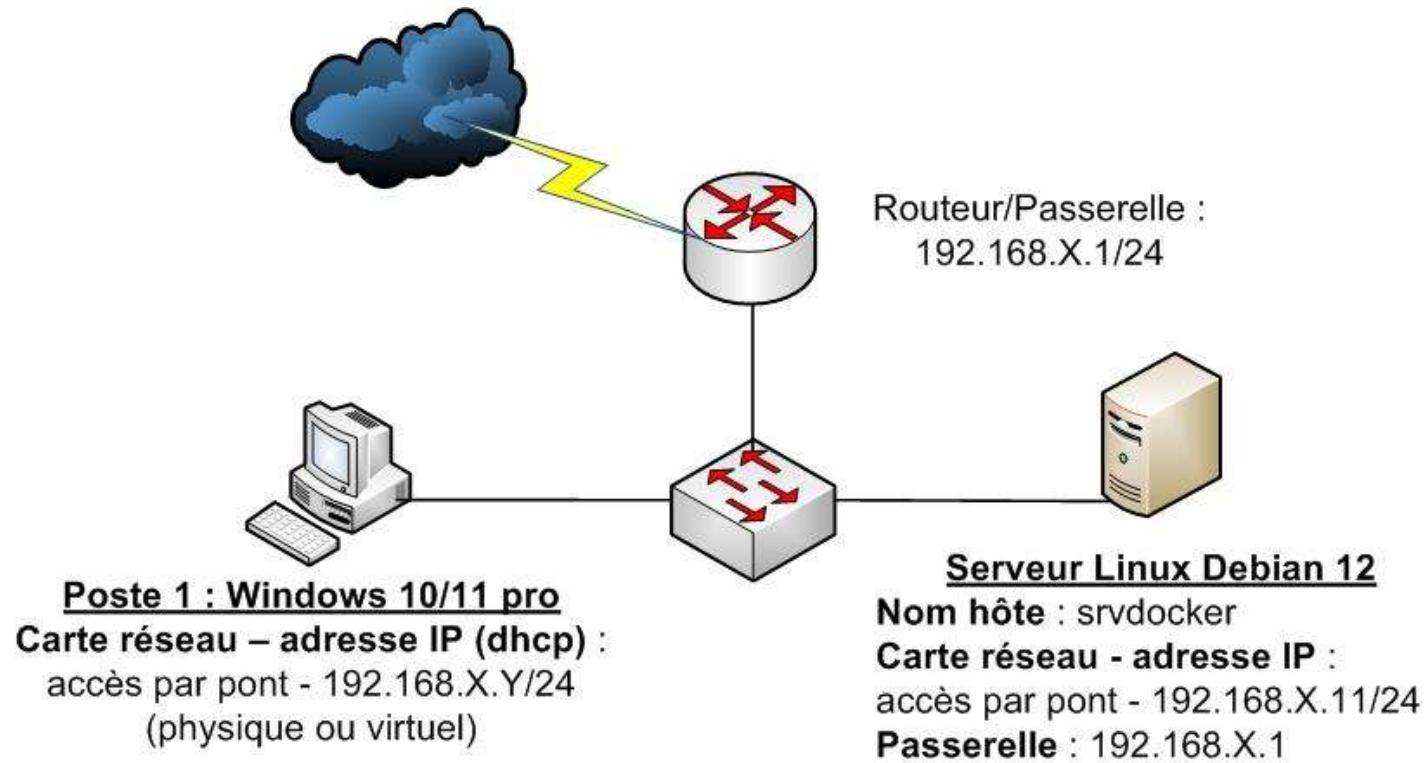
52. Réutilisez la commande trouvée à la **question n°48** et essayez d'accéder au site web. Que constatez-vous ?

53. Maintenant, allez dans le répertoire « **/usr/share/nginx/html** », listez son contenu et affichez le contenu du fichier « **index.html** » qui s'y trouve. Indiquez la procédure pas à pas que vous avez suivie. Que concluez-vous ?

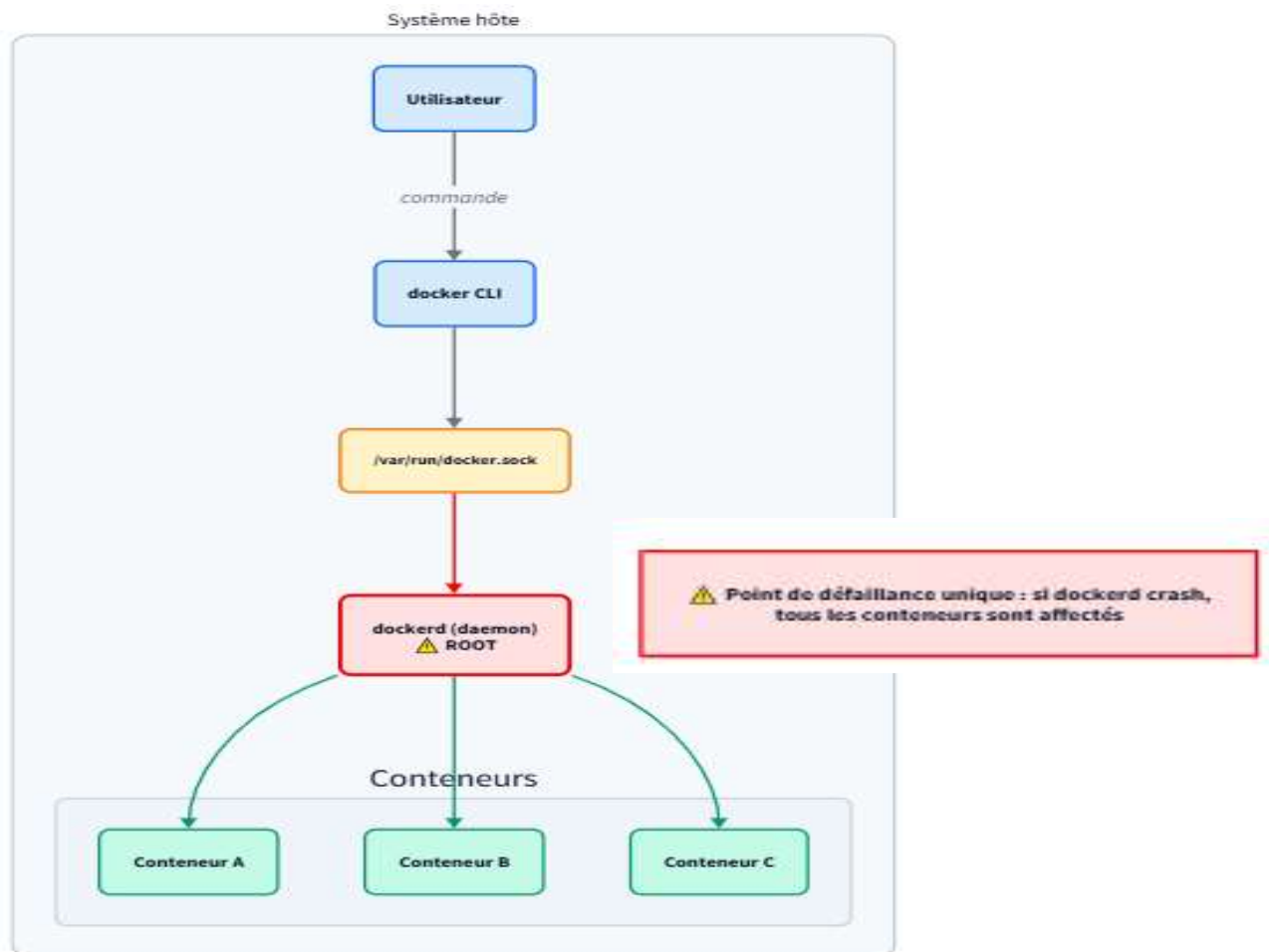
Bon travail !!!

ANNEXES

Document 1 : Architecture réseau à réaliser



Document 2 : L'architecture client-serveur de Docker



Document 3 : Les commandes essentielles pour manipuler des conteneurs via « Docker »

Action	Syntaxe des commandes
Afficher la version des composants logiciels de Docker : Engine, Containerd, ...	<code>docker --version</code>
Télécharger une image Docker et une version spécifique (étiquette/tag) depuis le site officiel Docker Hub.	<code>docker pull IMAGE_NAME:TAG (étiquette/version)</code>
Lister les images locales	<code>docker images</code> ou <code>docker image ls</code>
Supprimer une image	<code>docker rmi ID_IMAGE</code> ou <code>docker image rm ID_IMAGE</code>
Lister les conteneurs actifs et arrêtés (option -a)	<code>docker ps</code> ou <code>docker ps -a</code>
Chercher un conteneur	<code>docker search CONTAINER_NAME</code>
Créer un conteneur à partir d'une image. L'option -d permet de lancer le conteneur en mode détaché. L'option -it permet d'ouvrir un terminal interactif dans le conteneur.	<code>docker run -it --name CONTAINER_NAME OR IMAGE_NAME [COMMAND]</code>

Démarrer un conteneur	<code>docker start CONTAINER_NAME_OR_ID</code>
Arrêter un conteneur	<code>docker stop CONTAINER_NAME_OR_ID</code>
Mettre un conteneur sur pause Attention !!! <code>docker restart CONTAINER_NAME</code> , réinitialise le conteneur	<code>docker pause/unpause CONTAINER_NAME</code>
Supprimer un conteneur	<code>docker rm CONTAINER_NAME_OR_ID</code>
Voir les logs d'un conteneur	<code>docker logs CONTAINER_NAME_OR_ID</code>
Exécuter une commande dans un conteneur	<code>docker exec -it CONTAINER_NAME_OR_ID COMMAND</code>
Inspecter un conteneur	<code>docker inspect CONTAINER_NAME_OR_ID</code>
Obtenir des informations « réseau » d'un conteneur	<code>Docker network inspect bridge</code>
Copier des fichiers	<code>docker cp</code>
Nettoyer les conteneurs inutiles	<code>docker container prune</code>
Permet de quitter un conteneur	<code>exit</code>
Renommer un conteneur	<code>docker rename CONTAINER_OLD_NAME CONTAINER_NEW_NAME</code>

Document 4 : Méthodes de création de conteneurs et la plateforme d'images Docker Hub

Docker offre **3 méthodes** pour **créer des conteneurs** :

- **utiliser** une **image existante**,
- ou **construire votre propre image** avec un **Dockerfile**,
- ou **utiliser Docker Compose pour** des applications complexes, **multi-conteneurs**.

Trois méthodes de création de conteneurs

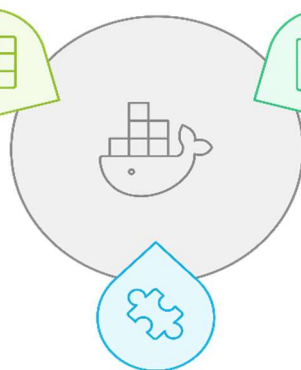
1) Image existante

Image préconstruite pour un déploiement rapide



2) Dockerfile

Un script définissant les instructions du conteneur

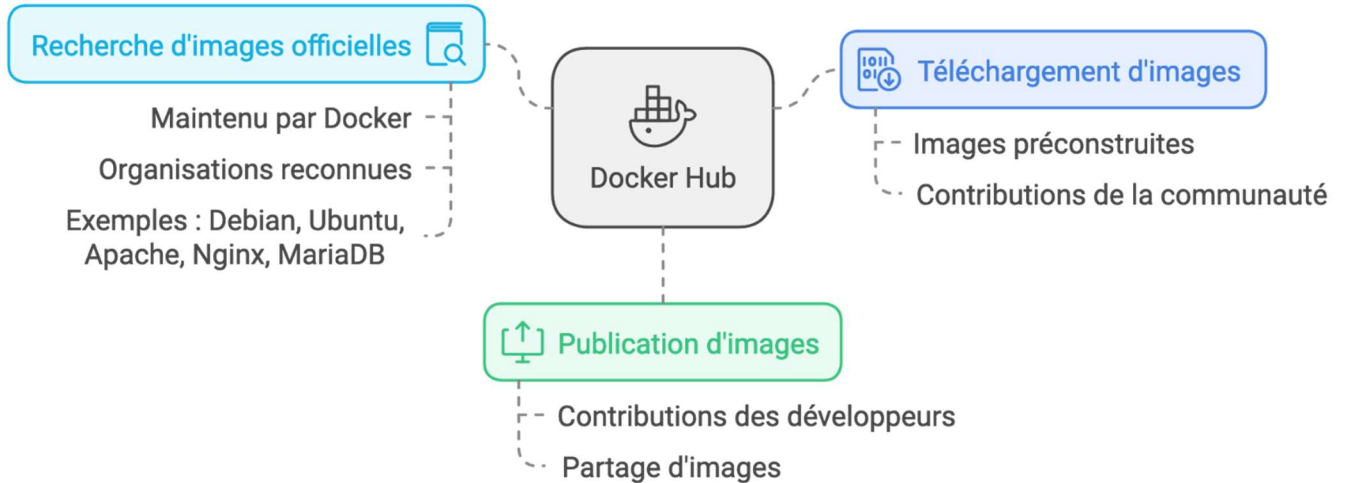


3) Docker Compose

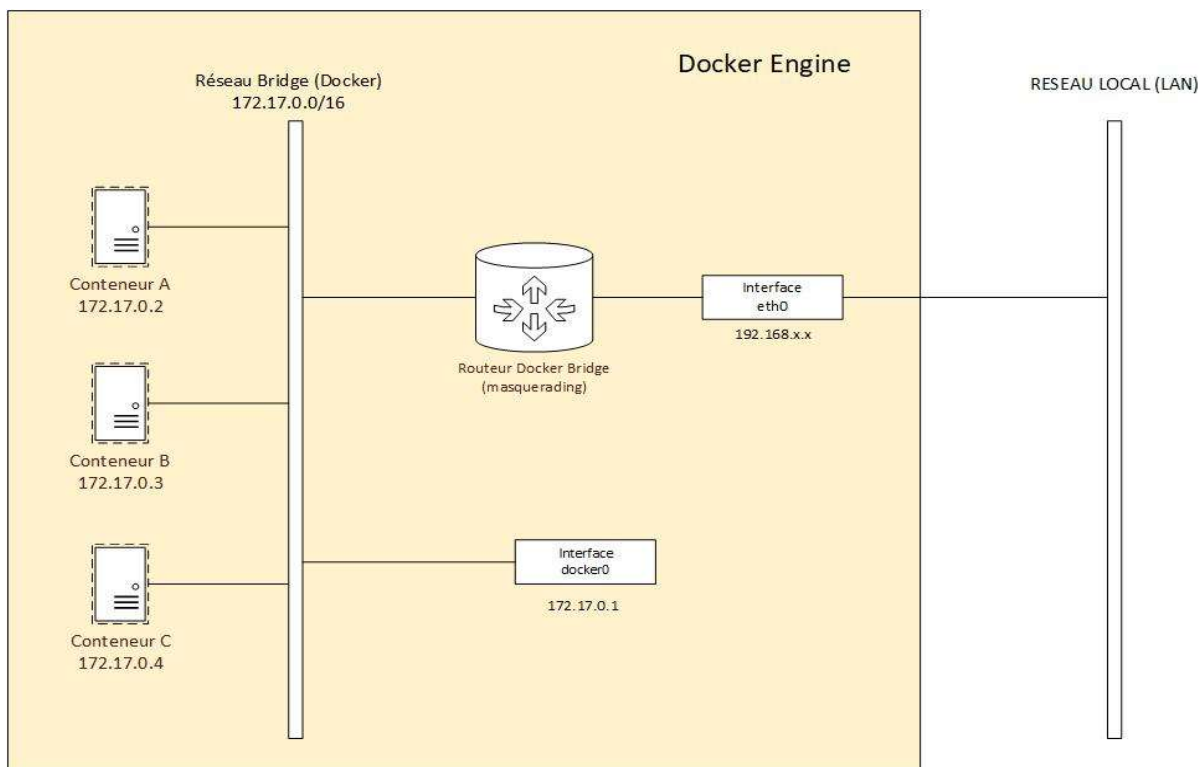
Outil de gestion des applications multi-conteneurs

Méthodes de création de conteneurs Docker

Plateforme d'images « Docker Hub »

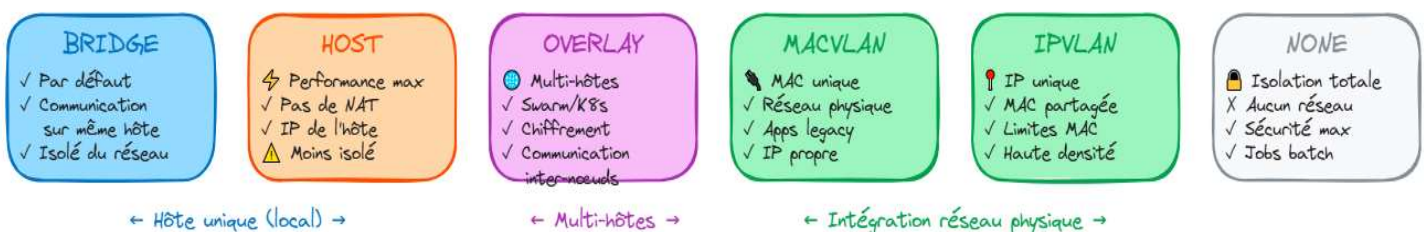


Document 5 : Représentation du réseau interne de Docker Engine



Document 6 : Les 6 types (drivers) de réseaux Docker

Docker propose **6 drivers réseau** natifs, chacun adapté à un cas d'usage spécifique. Le choix du bon réseau impacte directement la **sécurité**, les **performances** et la **facilité de maintenance** de vos applications conteneurisées.



Les **conteneurs Docker sont isolés par défaut** : ils ne peuvent pas communiquer entre eux ni avec l'extérieur sans configuration réseau. Docker propose **6 types de réseaux** pour répondre à tous les scénarios : du **développement local** à la **production distribuée** :

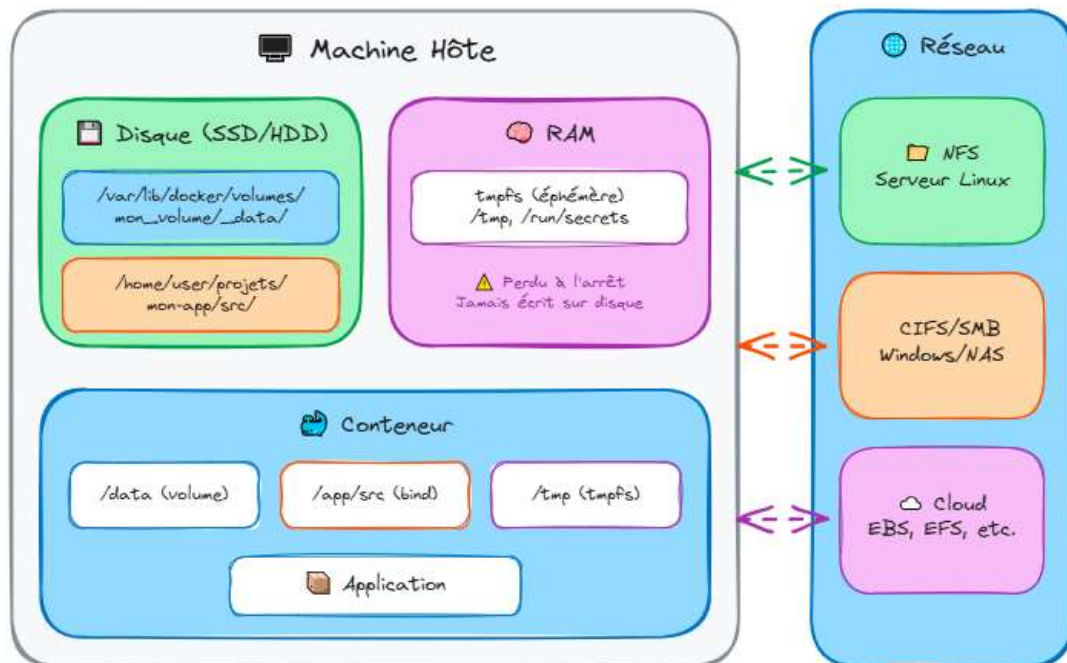
- **Bridge** : le réseau par défaut, idéal pour connecter des conteneurs sur un même hôte,
- **Host** : performances maximales en partageant le réseau de l'hôte,
- **Overlay** : communication entre conteneurs sur différents serveurs (Swarm/Kubernetes),
- **Macvlan/Ipvlan** : intégration directe au réseau physique (applications legacy)
- **None** : isolation totale pour les cas spéciaux de sécurité,

Voici les **bonnes pratiques et les règles à suivre** pour garantir la sécurité, dépannage et optimisation réseau :

- **Développement local** → Bridge (par défaut)
- **Production mono-hôte** → Bridge personnalisé
- **Production multi-hôtes** → Overlay
- **Performance critique** → Host (avec précautions)
- **Intégration réseau existant** → Macvlan/Ipvlan

Document 7 : Trois types de montage de volumes et persistance des données sous Docker

Architecture de stockage Docker – Où sont stockées les données ?



Avant de détailler les différents types de volumes, comprenons **où Docker peut stocker vos données**. Cette compréhension est essentielle pour choisir le bon type de volume selon votre architecture.

Stockage local vs stockage distant

La distinction fondamentale à comprendre :

Stockage local (sur la machine hôte) :

- **Volumes standards** : Docker crée un espace dédié sur le disque de l'hôte (`/var/lib/docker/volumes/`)
- **Bind mounts** : vous choisissez un dossier existant sur l'hôte (ex: `/home/user/projet`)
- **tmpfs** : utilise la RAM de l'hôte (pas de persistance)

Stockage distant (sur le réseau) :

- **NFS** : partage de fichiers entre serveurs Linux
- **CIFS/SMB** : partage Windows/Samba
- **Plugins** : SSHFS, cloud (AWS EBS, Azure Disk, etc.) → volume de stockage « DaaS » (Data as a Service)

Observations : Pourquoi c'est important ?

En environnement mono-serveur, le stockage local suffit.

Mais dès que vous avez **plusieurs serveurs Docker** (cluster Swarm, [Kubernetes](#)), les conteneurs peuvent migrer d'un hôte à l'autre. Sans stockage distant, les données restent sur l'ancien serveur et le conteneur redémarre à vide sur le nouveau.

Par analogie :

- Le **stockage local**, c'est comme *avoir vos fichiers sur le disque dur de votre PC*.
- Le **stockage distant**, c'est comme *avoir vos fichiers sur le réseau ou le CLOUD* : accessibles depuis n'importe quel appareil.

Les 3 types de montage de volumes sous Docker

