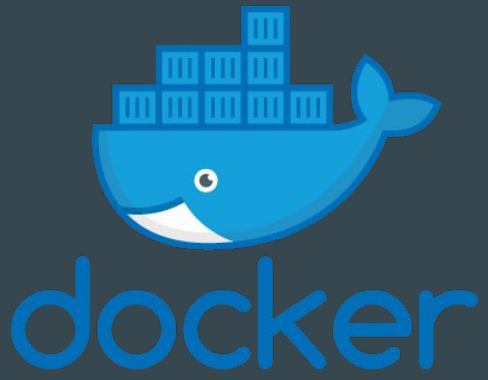


Fondamentaux de Docker

●●●

Enseigné par Madjid TAOUALIT

Développeur Full Stack, Ingénieur **DevOps** et Intervenant chez des écoles.

De quoi allons-nous parler ?

1. Courte introduction à Docker

- Qu'est-ce que **Docker** ?
- Comprendre **Docker**
- Comment cela se compare-t-il aux machines virtuelles

2. Cours pratique/tutoriel

- Installation du menu **Docker**
- Centre **Docker**
- Créer/télécharger une image
- Création d'un conteneur
- Mise à jour de l'application dans le conteneur
- Quelques commandes utiles pour les conteneurs
- **Dockerfile** – Image personnalisée

3. Suppléments

- Avantages de **Docker**
- Plus d'information

1.1. Qu’est-ce que Docker ?

- **Docker** est une plate-forme permettant aux développeurs et aux administrateurs système de développer, de livrer et d'exécuter des applications. **Docker** permet d'assembler rapidement des applications à partir de composants et élimine les frictions qui peuvent survenir lors de l'expédition du code. **Docker** nous permet de tester et de déployer le code en production le plus rapidement possible.

- **Docker** permet d'exécuter presque toutes les applications isolées en toute sécurité dans un conteneur. L'isolation et la sécurité nous permettent d'exécuter plusieurs conteneurs simultanément sur notre hôte.

1.2. Qu’est-ce que le moteur Docker ?

Docker Engine est une application client-serveur avec ces composants principaux :

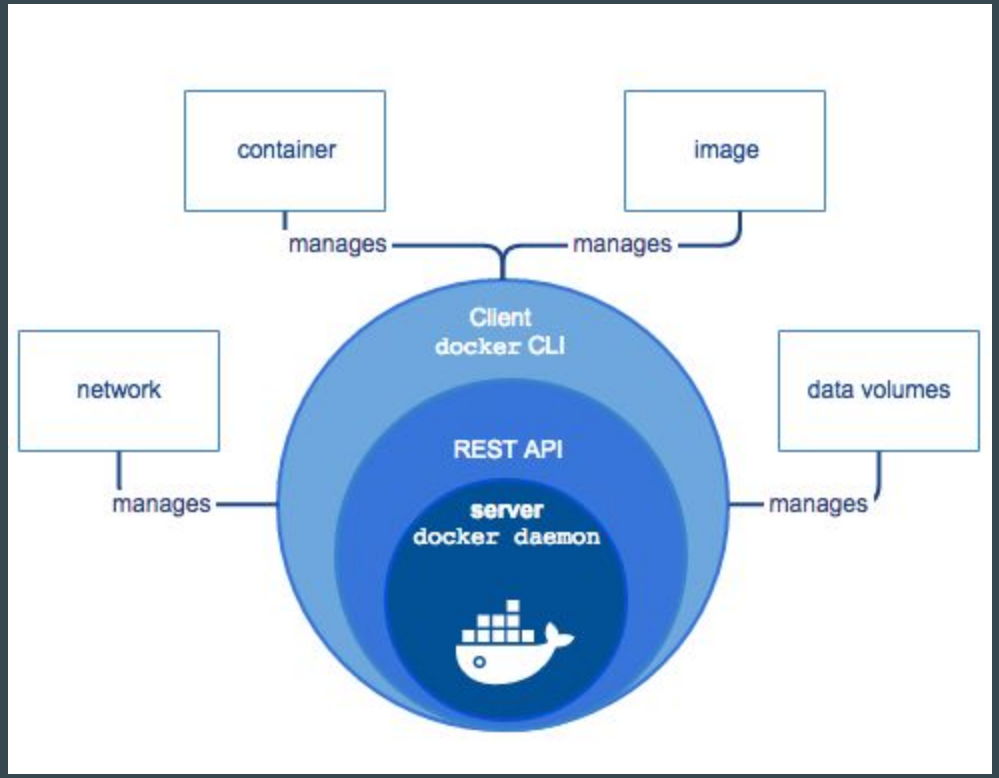
- Un serveur (processus **démon**).
- Une **API REST** qui spécifie les interfaces que les programmes peuvent utiliser pour communiquer avec le démon
- Un client d'interface de ligne de commande (**CLI**).

La **CLI** utilise l'**API REST** pour contrôler ou interagir avec le démon via des scripts ou des commandes **CLI** directes.

Le **démon** crée et gère les objets **Docker** , tels que les images, les conteneurs, les réseaux, les volumes de données, etc.

L'**utilisateur** n'interagit pas directement avec le démon, mais via le client **Docker** .

Le client, sous la forme du binaire **Docker** , est la principale interface utilisateur de **Docker** . Il accepte les commandes de l'utilisateur et communique dans les deux sens avec un démon **Docker** .


container
image
manages
manages
Client
docker CLI
network
data volumes
REST API
manages
server
docker daemon
manages

1.3. À l'intérieur de Docker

Pour comprendre les composants internes de **Docker** , on doit connaître les trois ressources :

Images Docker

Une image Docker est un modèle en lecture seule. Par exemple, une image peut contenir un système d'exploitation **CentOS** avec **Apache** et une application Web installée. Les images sont utilisées pour créer des conteneurs **Docker** . Les images **Docker** sont le composant de construction de **Docker** .

Registres Docker

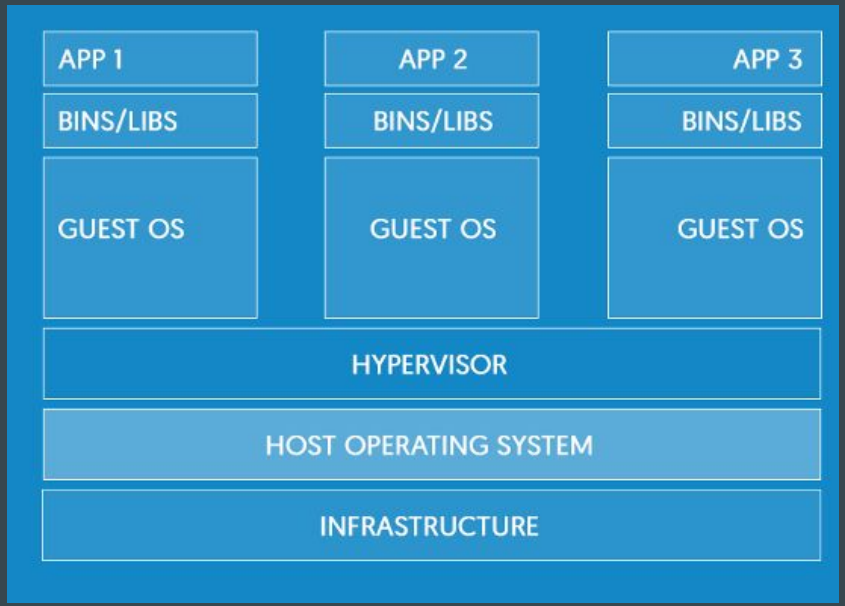
Les registres **Docker** contiennent des images. Il s'agit de magasins publics ou privés (Data Stores) à partir desquels on télécharge des images. Le registre **Docker** public est fourni avec le **Docker Hub** .

Conteneurs Docker

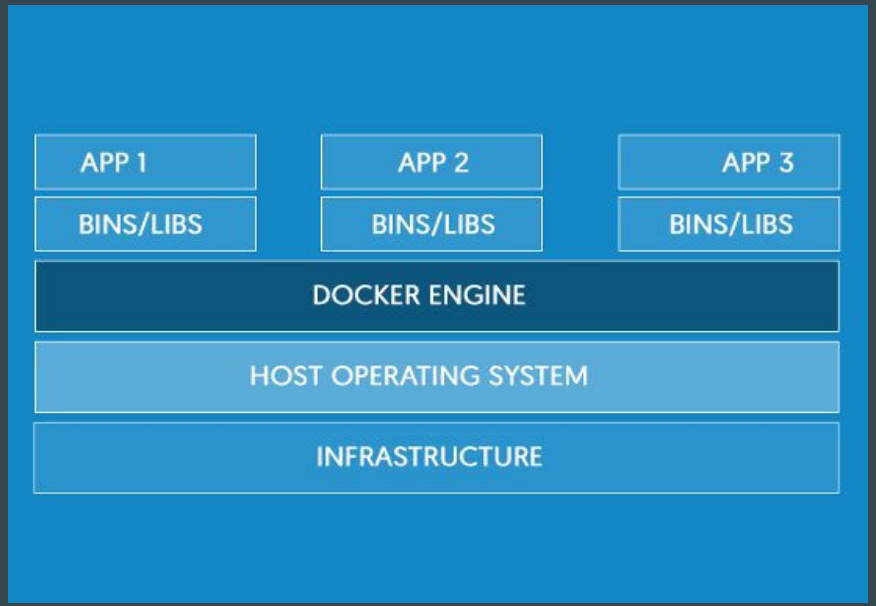
Les conteneurs **Docker** sont similaires à un répertoire. Un conteneur **Docker** contient tout ce qui est nécessaire à l'exécution d'une application. Chaque conteneur est créé à partir d'une image **Docker** . Les conteneurs **Docker** peuvent être exécutés, démarrés, arrêtés, déplacés et supprimés. Chaque conteneur est une plateforme applicative isolée et sécurisée. Les conteneurs **Docker** sont le composant d'exécution de **Docker** .

1.4. Conteneurs vs machines virtuelles

Architecture des machines virtuelles



Architecture de Docker



1.5. Centre Docker (Docker Hub)

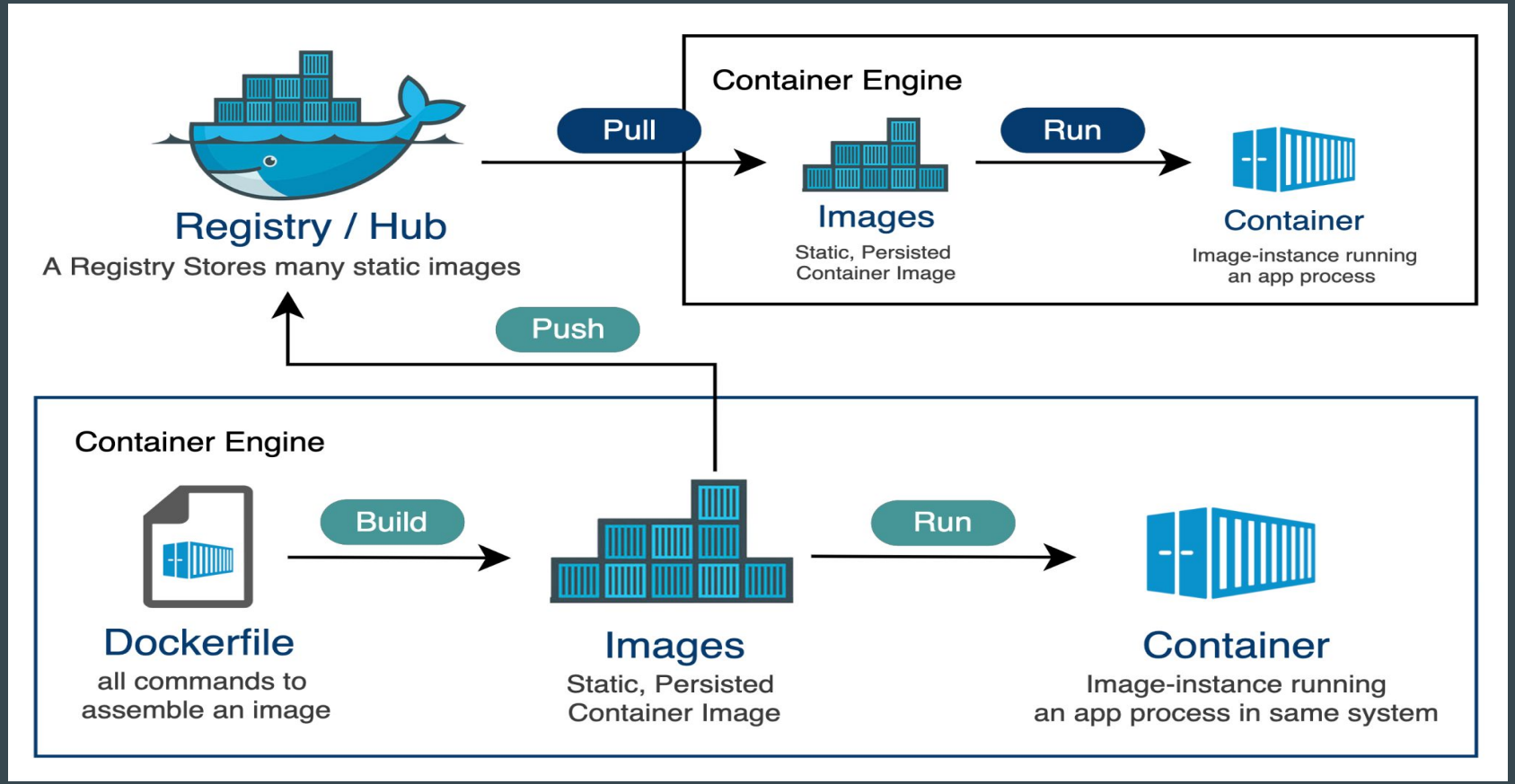
Qu'est-ce que Docker Hub ?

Docker Hub est un service de registre basé sur le cloud qui nous permet de créer des liens vers des référentiels de code, de créer nos images et de les tester, de stocker les images poussées manuellement afin qu'on puisse les déployer sur vos hôtes.

Docker Hub fournit les fonctionnalités principales suivantes :

- **Dépôts d'images :** Rechercher, gérer, transférer et extraire des images à partir de bibliothèques d'images communautaires, officielles et privées.
- **Constructions automatisées :** Créer automatiquement de nouvelles images lorsque nous apportons des modifications à un référentiel de code source.
- **Webhooks :** Fonctionnalité des builds automatisés, les Webhooks nous permettent de déclencher des actions après une transmission réussie vers un référentiel.
- **Organisations :** Créer des groupes de travail pour gérer l'accès aux référentiels d'images.
- **Intégration de GitHub et Bitbucket :** Ajouter le Hub et nos images Docker à nos flux de travail actuels.

Accessible sur le site Web https://hub.docker.com/


Container Engine
Pull
Run
Registry / Hub
A Registry Stores many static images
Images
Static, Persisted
Container Image
Container
Image-instance running
an app process
Push
Container Engine
Build
Run
Dockerfile
all commands to
assemble an image
Images
Static, Persisted
Container Image
Container
Image-instance running
an app process in same system

2.1. Installation de Docker (1)

Docker Engine est pris en charge sur **Linux**, **Cloud**, **Windows** et **OS X**.

Les instructions d'installation sont disponibles sur https://docs.docker.com/engine/installation/

CentOS Linux

- Créer un référentiel comme indiqué sur le site Web ci-dessus

```
# yum update & yum install docker-engine
# systemctl start docker && systemctl enable docker
```

Installation générale de Linux (devrait fonctionner dans la plupart des distributions **Linux**)

```
# curl -fsSL https://get.docker.com/ | sh
# systemctl start docker && systemctl enable docker
```

2.1. Installation de Docker (2)

Test d'installation

```
# docker image ls
# docker container ls
```

Donner accès à un autre utilisateur que root

```
# usermod -a -G docker <user>
```

(Connectez-vous avec l'utilisateur - si l'utilisateur est déjà connecté, il doit d'abord se déconnecter)

2.2. Créer/télécharger une image

Télécharger une image

```
# docker image pull <image_name_like_ind_docker_registry>
# docker image ls
```

Images multiples

```
# docker image pull <other_image_name>
# docker image ls
```

Plusieurs versions de la même image

```
# docker image pull <image_name>:<version>
# docker image ls
```

2.3. Création d’un conteneur (1)

Exécuter un conteneur

docker image ls
docker container run <image_name> (il sera exécuté en foreground)
docker container ls (-a)

Exécuter un conteneur en arrière-plan

docker container run -ti -d --name my_webserver httpd
docker container ls (-a)
- Essayez d'ouvrir http://container_ip

2.3. Création d’un conteneur (2)

Exposer les ports pour l'accès

```
# docker container run -ti -d -p 80:80 --name my_webserver httpd
```

Monter un point de montage externe à l'intérieur du conteneur

```
# docker container run -ti --name servidorweb -v /opt/external_mount/code/:/var/www/html:ro -d httpd
```

Accéder à la ligne de commande du conteneur

```
# docker container exec -ti my_webserver bash
```

2.4. Mise à jour d’un conteneur (1)

Enregistrer le conteneur modifié dans une nouvelle image

(Modifier le contenu du site Web)

```
# docker container commit docker_demo_apache docker_demo_apache:v2
# docker image ls
```

Exporter l'image vers un fichier

```
# docker image save -o image_save.tar <image_name>
```

Exporter le conteneur vers un fichier

```
# docker container export -o /home/container_save.tar <container_name>
```
(perd l'historique, aplatit l'image)

2.4. Mise à jour d’un conteneur (2)

Exécuter le conteneur avec la nouvelle version

```
# docker container stop <container>
# docker container rm <container>
# docker container run -ti -d -p 80:80 --name my_webserver docker_demo_apache:v2
```

Charger une image enregistrée

```
# docker image load -i <container_file>
```

2.5. Autres commandes Docker

Événements

```
# docker events
```

Inspecter le conteneur

```
# docker container inspect my_webserver
```

Journaux - Conteneur

```
# docker container logs my_webserver
```

Histoire - Image

```
# docker image history docker_demo_web:v2
```

2.6. Dockerfile

```
FROM centos:7
LABEL Madjid TAOUALIT <taoualitmadjid2@gmail.com>

RUN yum -y update
RUN yum install -y curl vim net-tools httpd git
RUN yum clean all

RUN git clone https://github.com/luisnabais/DOCKER-DEMO.git /var/www/html/

EXPOSE 80 443

COPY run-httpd.sh /run-httpd.sh
RUN chmod -v +x /run-httpd.sh

CMD ["/run-httpd.sh"]t
```

Création d'une image à partir de Dockerfile

```
# docker image build -t docker_demo_apache:v1
```

Conteneur en cours d'exécution

```
# docker container run -ti -d --name docker_demo_apache docker_demo_apache:v1
```

3.1. Avantages de Docker (1)

Avantages techniques

Légèreté

Les conteneurs exécutés sur une seule machine partagent le même noyau de système d'exploitation ; ils démarrent instantanément et utilisent moins de **RAM**.

Les images sont construites à partir de systèmes de fichiers en couches et partagent des fichiers communs, ce qui rend l'utilisation du disque et les téléchargements d'images beaucoup plus efficaces.

Ouverture

Les conteneurs **Docker** sont basés sur des normes ouvertes, permettant aux conteneurs de s'exécuter sur toutes les principales distributions **Linux**, **Microsoft Windows**, **Apple macOS** – et sur n'importe quelle infrastructure.

Sécurité

Les conteneurs isolent les applications les unes des autres et de l'infrastructure sous-jacente, tout en fournissant une couche de protection supplémentaire pour l'application.

3.1. Avantages de Docker (2)

Avantages pour les développeurs

Fini le « Ça fonctionne pas sur ma machine »

Développement accéléré

On ne perd plus de temps à configurer des environnements de développement, à créer de nouvelles instances et à faire des copies du code de production à exécuter localement.

Avec **Docker**, on prends simplement des copies de notre environnement en direct et les exécuter sur n'importe quel nouveau point de terminaison exécutant un moteur **Docker**.

Éliminer les incohérences de l'environnement

Conditionner une application dans un conteneur avec ses configurations et dépendances garantit que l'application fonctionnera toujours comme prévu dans n'importe quel environnement : localement, sur une autre machine, en test ou en production.

Ne plus se soucier de devoir installer les mêmes configurations dans différents environnements.

3.1. Avantages de Docker (3)

Avantages pour l’équipe des opérations

Moins d’effort pour maintenir les environnements

La création d'un environnement est automatisée. Par conséquent, moins d’actions manuelles sont nécessaires. Cela réduit les risques et augmente la fiabilité.

La maintenance manuelle d'un environnement cohérent sur plusieurs serveurs est sujette aux erreurs et peut s'avérer un cauchemar. Avec **Docker**, il est facile de créer plusieurs instances d’un environnement, car il suffit d’exécuter l’image sur les serveurs. De cette façon, il est facile d'ajouter des nœuds supplémentaires à un cluster et d'évoluer horizontalement.

Mise à jour facile

L'utilisation d'approches traditionnelles pour configurer un environnement (comme les scripts d'installation) pose problème lorsqu'il existe déjà un environnement existant. Considérer le chemin de mise à jour dans le script peut être une tâche très complexe (par exemple vérifier l'existence de fichiers et nettoyer les fichiers inutilisés). Avec **Docker** nous n'avons pas besoin de prendre en compte un environnement existant (sauf pour la base de données). Nous arrêtons simplement le conteneur en cours d'exécution et démarrons le nouveau conteneur mis à jour. Cela simplifie la configuration car nous commençons toujours avec un environnement vide.

3.2. Où puis-je obtenir plus d'informations ?

Documentation **Docker** : https://docs.docker.com

Comprendre **Docker** : https://docs.docker.com/engine/understanding-docker/

Dockerfile

Référence du fichier **Docker** : https://docs.docker.com/engine/reference/builder/

Exemples de fichiers **Dockerfile CentOS** : https://docs.docker.com/engine/reference/builder/

ENTRYPOINT contre **ADD** : https://www.ctl.io/developers/blog/post/dockerfile-entrypoint-vs-cmd/

AJOUTER ou **COPIER** : https://www.ctl.io/developers/blog/post/dockerfile-add-vs-copy/