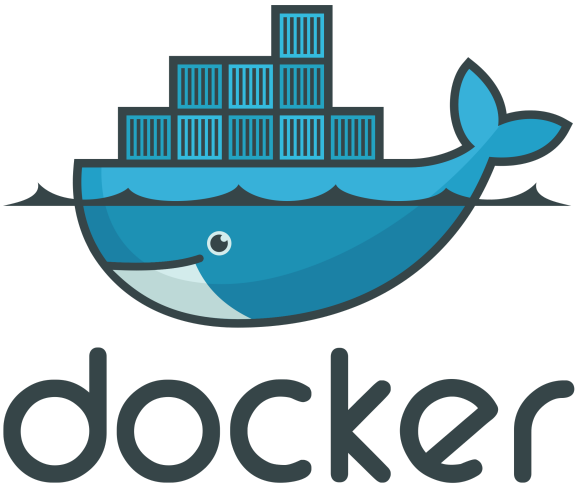
# Introduction à Docker

KIT APPRENANT



# Introduction à




## Objectif global

- Se familiariser avec Docker
- Mettre en place un environnement de production
- Comprendre les différents niveaux d'encapsulation d'environnement.
- Réviser l'utilisation des lignes de commandes

## Compétences

- Mise en place de l'environnement Docker
- Construire et publier des images Docker
- Mettre en place une stack avec docker-compose

## Démarche pédagogiques

- Installer Docker et comprendre les **principes de base** de la "conteneurisation"
- **Lancer et interagir** avec des containers
- Utiliser docker pour rendre vos projets facilement déployables, partageables, …
- **Créer vos propres images** Docker

- Utiliser des **images de Docker Hub**
- Mettre en place une **stack** et la lancer avec **Docker-Compose**

# MODALITÉS

## Durée

Durée du projet : 3 jours.

## Formateur(s)

Florian Dadouchi

# Introduction à Docker

KIT APPRENANT

## Références utiles tout au long du projet :

- ➔ Docker.com
- ➔ Official Documentation
- ➔ Docker Cheat Sheet

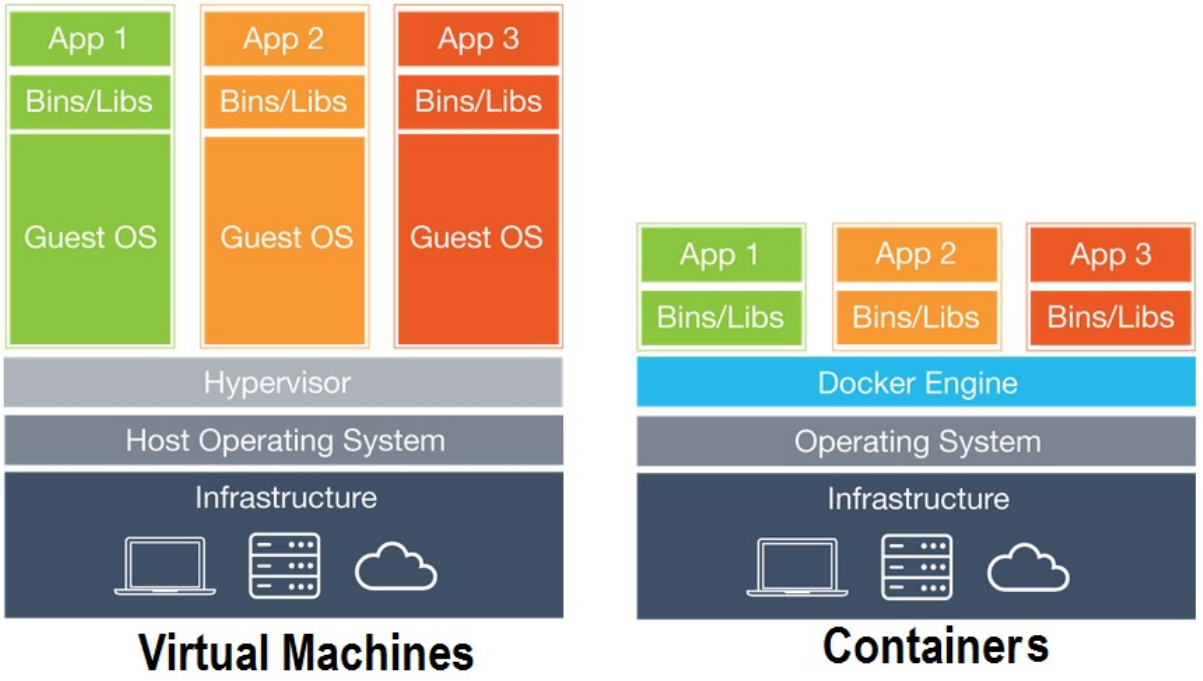

Un jeune étudiant du Campus Numérique ne connaissait pas Docker

### Qu'est-ce que Docker?

**Docker** est un outil **open source** permettant de **"packager" des applications et leur dépendances dans des fichiers images**. Contrairement aux machines virtuelles (Machine virtuelle — Wikipédia), **les images ne contiennent pas d'OS**, mais seulement les librairies nécessaires pour faire tourner l'application.

# Introduction à Docker

KIT APPRENANT




A partir de **fichiers image**, **Docker instancie des "Containers"** en les **isolants du système hôte** et en leur donnant  accès seulement à ce que l'on souhaite.

> **Un container est un processus isolé,**
> **accédant uniquement aux ressources qui lui ont été autorisées**.

L'isolation de processus existe depuis longtemps sous Linux, mais fait appelle à des **notions avancées pas toujours faciles à comprendre** et à mettre en œuvre.

La force de docker est d'**avoir démocratisé l'utilisation de ces mécanismes** et de permettre à la communauté d'**échanger et de réutiliser des milliers d'images au travers d'un Hub**.

Nous allons maintenant débuter le projet par l'installation de Docker, l'inscription sur Docker Hub, la vérification de l'installation et l'apprentissage des commandes basiques de Docker en CLI.

---

## I. Installation de Docker

- Commencez par installer Docker sur votre machine → Get Docker

- Pour les utilisateurs de Linux et Mac vous ne devriez pas avoir de problème d'installation. Pour les utilisateurs de Windows, l'installation peut être un brin fastidieuse → Install Docker Desktop on Windows).

**Etape post-installation optionnelle :**

Après l'installation de Docker (sous Linux) il est possible de vous débrouiller pour ne plus avoir à taper "**sudo**" à chaque commande Docker en créant un groupe docker et en rajoutant l'utilisateur **non root** à ce groupe → Linux post-installation steps for Docker Engine

C'est une étape que vous retrouverez régulièrement mais qui pose de **sérieux problèmes de sécurité** ! → Linux users: running docker without sudo is dangerous | by suixo | Medium

---

## II. Création un compte sur Docker Hub

- Nous allons maintenant créer un compte sur Docker Hub, l'équivalent de GitHub pour les images Docker (https://hub.docker.com/).
- Une fois enregistré sur Docker Hub vous aurez accès aux images Docker publiques. Vous pourrez également sauvegarder vos propres images publiques ou privées (une image privée autorisée avec un compte gratuit).

**Ressource** : https://docs.docker.com/docker-hub/

---

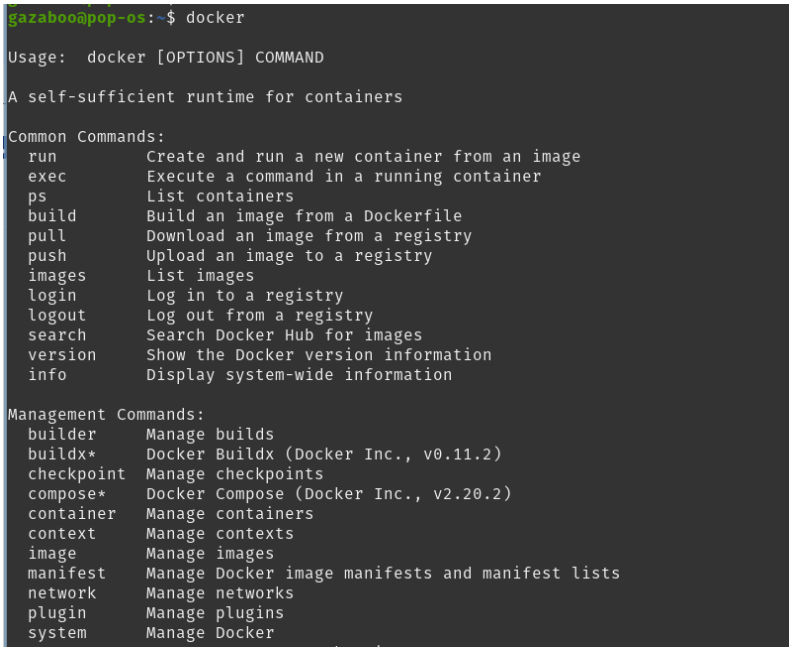
## III. Vérification de l'installation

Une fois l'installation terminée, lancer un **shell** (ou **cmd prompt** sous Windows).

Dans le terminal taper :

```
> docker
```

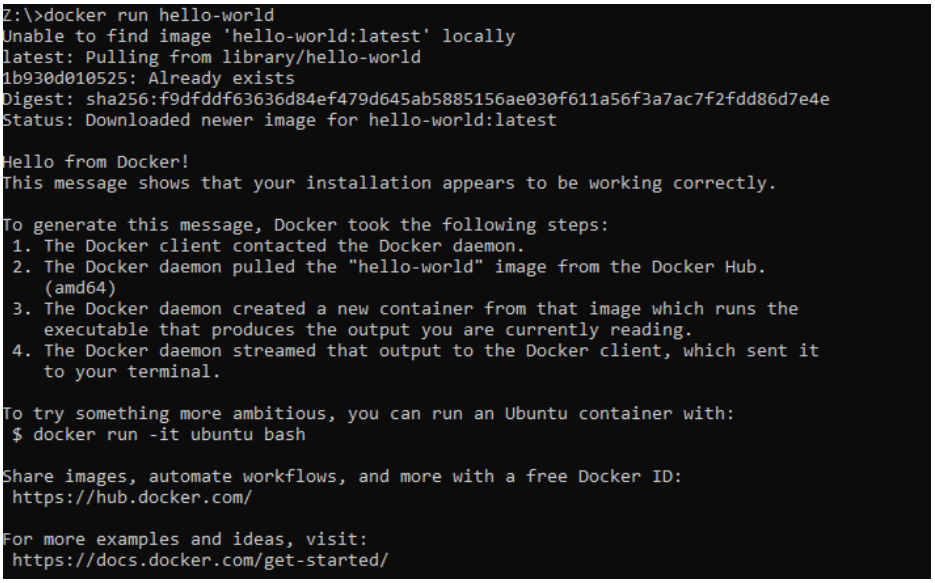
Vous devriez obtenir la sortie classique :




Vous allez maintenant **lancer votre premier container** via l'image hello-world :

```
> docker run hello-world
```

"**Hello-world**" est une **image** toute simple qui affiche un message **permettant de valider que Docker est bien installé** et que tout fonctionne sur votre système. **Si vous avez un message d'erreur, contactez le formateur.**



## IV. Les commandes de Docker

La CLI (Command Line Interface) Docker contient **toutes les commandes permettant d'interagir avec nos containers** ainsi que des commandes d'administration du système Docker (Management Commands).

Vous retrouverez l'ensemble des commandes Docker ici :

- Use the Docker command line

Quelques commandes de base ici :

- Docker Commands Tutorial | Top 15 Docker Commands | Edureka

Mais voici quand même quelques commandes générales avec lesquelles nous allons jouer ici :

```
> docker <command> --help
```

```
> docker --version
```

```
> docker info
```

**Liste les images locales**

```
> docker image ls
```

ou

```
> docker images
```

**Questions :**

1. Quel est l'identifiant (ID) de l'image "Hello-world"?
2. Quelle est sa taille?
3. Quel est son Tag?

**Note:**
Les Identifiants sont uniques et **permettent d'identifier une ressource** quel que soit son

type (Image, container, network, stack…). On peut utiliser les **3 premiers caractères** (ou 4 ou plus) pour le référencer.

### Lancer un container

```
> docker run <image ID>
```

### Lister les containers “running”

```
> docker container ls
```

ou

```
> docker ps
```

### Liste tous les containers

```
> docker container ls -a
```

(avec a pour all) ou

```
> docker ps -a
```

### Questions :

1. Quels sont les identifiants de vos containers?
2. Quels sont leurs Name?
3. Quel est leur Status?

### Relancer un container stoppé

```
> docker start <container ID>
```

Ou de manière interactive

```
> docker start -i <container ID>
```

### Supprimer tous les containers stoppés

```
> docker container prune
```

**Supprimer une image**

```
> docker image rm <image ID>
```

(rm pour remove) ou

```
> docker rmi <image ID>
```

(rmi pour remove image)

---

## V. Utiliser des images publiques

Le Hub docker contient des milliers d'images réutilisables dont certaines sont des **images de bases optimisées pour être réutilisées** comme point de départ pour créer nos propres images.

**Note :**
Une image très connue pour sa petite taille est basée sur **linux alpine**
**Linux Alpine** utilise **"apk"** comme package manager.

Nous allons commencer par aller chercher et exécuter une image Docker du nom de `gazaboo/webapp_campus` :

```
> docker run -p 5000:5000 gazaboo/webapp_campus:1.0
```

Puis allez vérifier que tout fonctionne bien sur http:/127.0.0.1:5000/

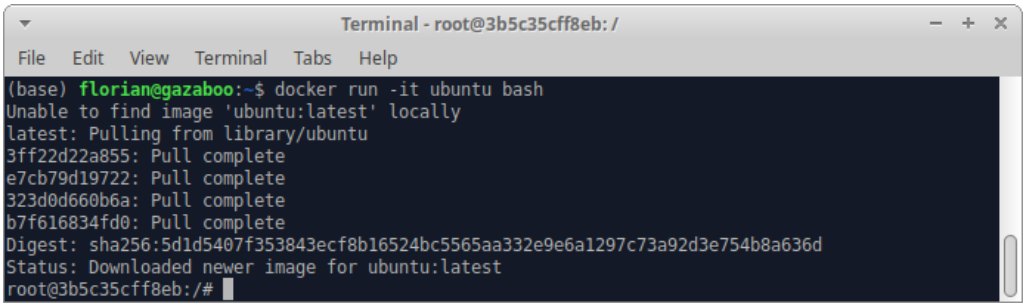
**Nous allons commencer par utiliser une image Ubuntu disponible ici** → ubuntu - Official Image | Docker Hub

Nous pourrions commencer par la télécharger en utilisant la commande pull :

```
> docker pull ubuntu
```

Mais nous n'en avons pas besoin. Nous pouvons directement lancer l'image Ubuntu en utilisant run :

```
> docker run -it ubuntu bash
```




Nous voyons que docker ne trouve pas l'image d'ubuntu en local (**"Unable to find image 'ubuntu:latest' locally"**) et va donc lancer un appel à **pull** automatiquement. Le rajout du mot clé **`bash`** permet de dire au conteneur quelle commande exécuter après avoir lancé le conteneur, dans notre cas : la commande bash. Les deux arguments **`-i`** et **`-t`** permettent d'obtenir une exécution interactive du container, et de demander d'émuler un pseudo terminal (tty). Vous pouvez essayer d'exécuter la commande sans l'un ou l'autre des arguments pour comprendre ce qu'il se passe. En cas de soucis (terminal bloqué!) allez voir cette ressource : [How to Run Docker Containers [run and exec]]

Vous êtes donc maintenant dans un terminal bash Linux classique, vivant dans un container!

**Jeu inoffensif (1) :**

Pour les gens sains d'esprit (par opposition à ceux qui veulent essayer le jeu de la mort), une fois à l'intérieur du bash dans le container, essayez des commandes quelques commandes classiques. Comme par exemple :

```
>> apt update
>> apt install figlet
>> figlet hello campus!
```

**Jeu inoffensif (2) :**

1. Quel est l'identifiant de votre container ubuntu. Quel est son ID, status et son nom?
2. Stoppez le container puis relancez le. Exécutez à nouveau la commande `figlet hello campus.` (https://docs.docker.com/engine/reference/commandline/start)

**Jeu inoffensif (3) :**

1. Exécutez la commande suivante :
   **`>> docker system prune`**
   Que fait elle ? Allez chercher la ressource d'explication.
   Vérifier le résultat avec un
   **`>> docker ps -a`**
   Les résultats sont-ils cohérents ?
2. Relancez un container ubuntu en appelant le bash de manière interactive et ré-exécutez la commande suivante :

```
>> figlet hello
```

Que se passe-t-il cette fois ?

3. Utilisez la commande `docker commit` (https://docs.docker.com/engine/reference/commandline/commit/) pour créer une nouvelle image dans laquelle le paquet figlet est déjà présent.
4. Quelle est la différence entre un container et son image ?

**Jeu (très) dangereux :**

**Une fois à l'intérieur du container** vous pouvez vous amuser à exécuter des commandes que vous ne devriez autrement **JAMAIS** utiliser dans un bash normal (**JAMAIS! JAMAIS! JAMAIS!**).

Exemple :

- `rm -rf usr/bin/`
  → suppression des fonctions bash. Testez un ls par exemple suite à cette commande.

- `rm -rf /`
  → suppression de l'ensemble du système de fichier

Faites attention à **bien vérifier que vous êtes bien à l'intérieur du container !** Sinon vous êtes foutus ! → https://kb.iu.edu/d/abeh

## VI. Dockerfile

Jusqu'à présent nous avons lancé des images existantes et interagit avec nos containers de manière interactive. Nous allons maintenant **créer nos propres images grâce à un Dockerfile**. Un fichier Dockerfile contient les **instructions nécessaire pour créer et modifier une image à partir d'une image de base** :

> Docker can **build images automatically by reading the instructions from a Dockerfile**. A Dockerfile is a text document that contains all the commands a user could call on the command line to assemble an image. Using docker build users can create an automated build that executes several command-line instructions in succession.
>
> **Site officiel Docker**
> https://docs.docker.com/engine/reference/builder/

### Premier Dockerfile avec commande CMD

Nous allons commencer par créer un Dockerfile extrêmement basique. Dans un **répertoire de travail vide** de votre choix, **créer un fichier nommé "Dockerfile"** et l'éditer comme suit :

```
FROM  alpine
LABEL maintainer="DockerGeeK"

CMD ["echo", "Conversation depuis l'intérieur de la baleine"]
```

CMD est la commande par défaut à lancer au démarrage du container
→ Dockerfile reference | Docker Docs

Vous pouvez maintenant créer l'image à partir de ce Dockerfile (en donnant un nom à l'image), en utilisant la commande suivante :

```
> docker build -t nom_image .
```

Et lancez cette image :

```
> docker run nom_image
```

Observe-t-on bien la sortie désirée ?

### Dockerfile avec commande ENTRYPOINT

```
FROM  alpine
LABEL maintainer="DockerGeeK"

RUN     apk update \
     && apk upgrade \
     && apk add figlet

ENTRYPOINT ["figlet"]
```

Vous pouvez une fois encore créer l'image à partir de ce Dockerfile :

```
> docker build -t <nom_image> .
```

ENTRYPOINT permet de configurer un container qui va alors fonctionner comme un exécutable. On peut donc exécuter le container, en lui passant un paramètre, de la manière suivante :

```
> docker run <nom_image> bonjour
```

```
> docker run <nom_image> inside la baleine
```

Voici quelques instructions utilisables pour créer un Dockerfile :

- FROM qui vous permet de définir l'image **source** ;
- RUN qui vous permet d'exécuter des **commandes** dans votre conteneur ;
- ADD qui vous permet **d'ajouter des fichiers** dans votre conteneur ;
- WORKDIR qui vous permet de définir votre **répertoire de travail** ;
- EXPOSE qui permet de définir les **ports d'écoute** par défaut ;
- VOLUME qui permet de définir les **volumes utilisables** ;
- CMD qui permet de définir la **commande par défaut** lors de l'exécution de vos conteneurs Docker.

Nous allons en servir un peu plus dans la partie suivante.

**Ressources :**

- Doc officielle : [Dockerfile reference | Docker Docs]

- Open Classrooms : Créez votre premier Dockerfile - Optimisez votre déploiement en créant des conteneurs avec Docker - OpenClassrooms
- Grafikart (vidéo) : Tutoriel vidéo Docker : Les dockerfile | Grafikart

---

## VII. Exemple d'utilisation de Docker avec un serveur NGINX

Nous allons lancer un container avec NGINX installé en mode background (ou détaché) et voir comment nous allons **"mapper" un port du container avec un port du Host.**

> **NGINX** est un logiciel libre de serveur Web (ou HTTP) ainsi qu'un proxy inverse écrit par Igor Sysoev [...] c'est depuis avril 2019, le serveur web le plus utilisé au monde [...].
> Wikipedia.

### Lancer le serveur NGINX et vérifier que l'on y a accès

```
> docker run -d nginx

(Note : le paramètre d est pour detached, la doc ici → Docker run reference )

> docker ps
> docker inspect <container ID>

(allez chercher l'adress IP (IP Address) et la noter)

> ping <IP Address>
> curl <IP address>
```

Par défaut, une adresse IP est associée à chaque container qui se connecte à un réseau Docker. L'adresse IP est assignée aléatoirement à partir des adresses IP disponibles sur le réseau (plus d'explications → Networking overview | Docker Docs)

Ouvrez un navigateur et vérifiez que vous tombez bien sur la page d'accueil du serveur NGinx en visitant → http://<IP address>:80

**Notes :**

**ping <ip address>** est une commande qui permet de vérifier si une adresse IP est accessible.

**curl <url>** est une commande qui envoie une requête HTTP sur une URL

Un **port de communication** permet de définir un point de terminaison unique sur une machine. Il permet **pour une seule adresse IP d'avoir plusieurs milliers de points de terminaisons** rendus uniques par leur **numéro de port**. Par convention **un serveur web "écoute" sur le port 80 pour le protocole HTTP et le port 443 pour le protocole HTTPS**.

*By default, the container is assigned an IP address for every Docker network it connects to. The IP address is assigned from the pool assigned to the network.*
*Site de Docker → Networking overview | Docker Docs*

## Mapping de ports

Une manière pratique de lancer un tel serveur est de **"mapper"** les ports du container aux ports physiques de la machine hôte du container :

```
> docker run -d -p 8080:80 nginx
```

**Note :**
L'option "**-p 8080:80**" permet de **"mapper"** le **port 8080 de votre machine** (ou docker est installé) sur le **port 80 du container**

Vous pouvez maintenant ouvrir un navigateur sur la page http://localhost:8080 et observer le résultat.

## Copie de fichiers de la machine vers le container

Nous allons changer la page HTML d'accueil renvoyée par NGinx lors d'une connexion. Par défaut Nginx va chercher les fichiers à servir dans `/usr/share/nginx/html`, vous pouvez trouver cette information dans la documentation mais aussi en analysant le contenu du fichier **`/etc/nginx/conf.d/default.conf`**

**Petit exercice :**
Afficher le contenu du fichier **`default.conf`**, et trouver l'endroit où le chemin par défaut est spécifié.

**Créez un répertoire "nginx"** à l'emplacement de votre choix sur votre PC et **créez un fichier index.html à l'intérieur** contenant un peu de HTML. Voici un exemple de Dockerfile récupérant le fichier index.html

```
FROM  nginx
LABEL maintainer="DockerGeeK"

COPY ./index.html /usr/share/nginx/html/

EXPOSE 80
```

Vérifiez maintenant que vous atterrissez sur la bonne page d'accueil !

**Petit exercice :**
Modifiez le contenu du fichier index.html et rechargez la page. Que se passe-t-il ? Pourquoi?

## Mapping de répertoires

Une manière simple de travailler avec des containers en phase de développement est de "mapper" des volumes (répertoires) entre notre machine physique et le système de fichier du container. C'est analogue à ce que nous avons fait en "mappant" des ports.

```
> docker run -d -p 8081:80 -v <path>:/usr/share/nginx/html nginx
```

**<path>** étant le path (absolu) vers le dossier contenant vos fichiers sur votre machine physique.

**Petit exercice :**
Modifiez le contenu du fichier index.html et rechargez la page. Que se passe-t-il ? Pourquoi?

Vous pouvez maintenant commencer à faire le ménage, stopper et effacer les containers.

```
> docker ps
> docker kill <ID1> <ID2> ...
> docker stop ...
> docker -p
```