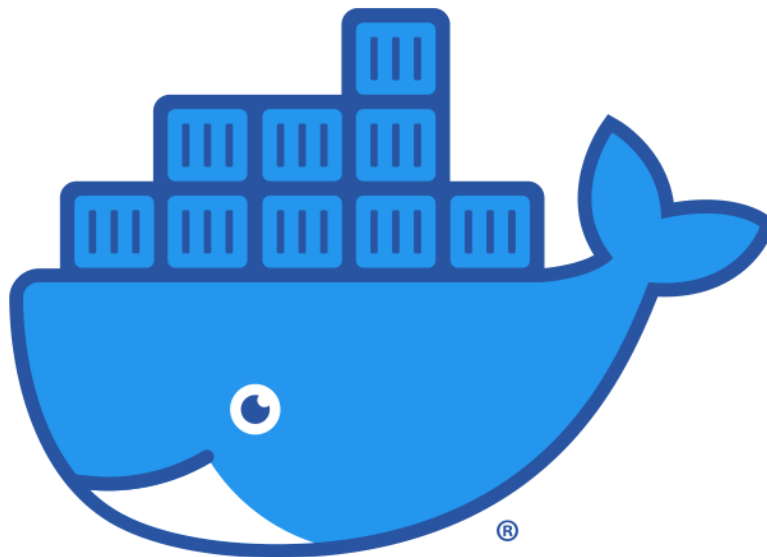


# Utiliser Docker



## TABLE DES MATIERES

<b><i>Section 1 But.....</i></b>	<b><i>3</i></b>
1.1 But.....	3
1.2 Liens Web.....	3
<b><i>Section 2 Principes.....</i></b>	<b><i>4</i></b>
2.1 Généralités.....	4
2.2 Les versions de Docker.....	5
<b><i>Section 3 Installer Docker.....</i></b>	<b><i>6</i></b>
<b><i>Section 4 Un premier conteneur Docker.....</i></b>	<b><i>7</i></b>
4.1 Lancer Docker Desktop.....	7
4.2 Exemple avec un serveur Nginx.....	7
<b><i>Section 5 Créer une image Docker élémentaire.....</i></b>	<b><i>9</i></b>
<b><i>Section 6 Image avec NodeJS.....</i></b>	<b><i>10</i></b>
6.1 But.....	10
6.2 Mise en œuvre.....	10
6.3 Effectuer une modification.....	11
6.4 Partager l'application sur Docker Hub.....	12
6.5 Essayer notre conteneur.....	13
6.6 Réaliser une persistance de la base de données.....	13
6.7 Autre type de persistance avec Bind Mounts.....	14
6.8 Application multi conteneurs.....	14
<b><i>Section 7 Utilisation de docker compose.....</i></b>	<b><i>17</i></b>
7.1 But.....	17
7.2 Utilisation de compose.....	17
<b><i>Section 8 Docker : les bonnes pratiques.....</i></b>	<b><i>19</i></b>

## Section 1 But

---

### 1.1 But

Cette initiation à Docker s'inscrit dans le module MS2D « Ingénierie et développement ».

En effet la plateforme d'intégration continue GitLab CI utilise ce support pour les tests.

Idem pour le déploiement continu Gitlab.

### 1.2 Liens Web

<https://openclassrooms.com/fr/courses/2035766-optimisez-votre-deploiement-en-creant-des-conteneurs-avec-docker>

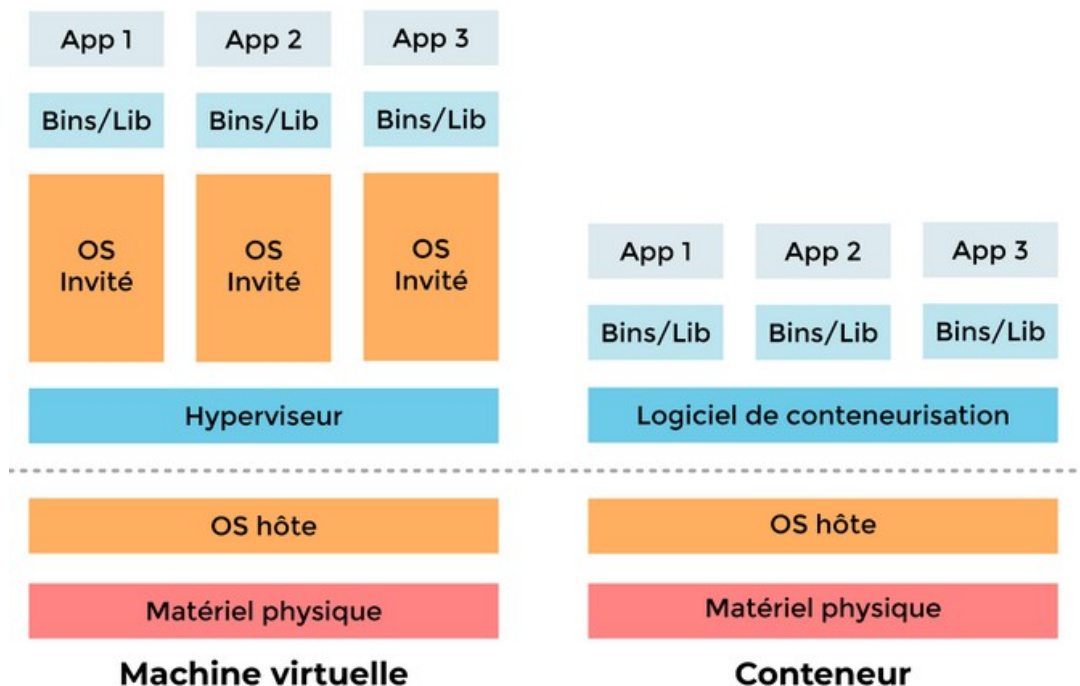
## Section 2 Principes

### 2.1 Généralités

Un conteneur d'application permet de contenir une application et le juste suffisant pour le faire tourner sur n'importe quelle machine de n'importe quel OS, pourvu que le système de conteneur installé le permette.

Une machine Virtuelle émule un OS sur une machine qui peut être d'un OS différent, l'environnement est alors lourd.

Le système de conteneur est une virtualisation légère. Il ne virtualise pas les ressources (clavier, gestion écran, disque ...) mais crée une isolation des processus. Le conteneur partage les ressources de l'OS hôte.



Comparaison entre les conteneurs et les machines virtuelles

LXC et OpenVZ sont des systèmes de conteneurs antérieurs à Docker.

Les conteneurs partagent entre eux le Kernel Linux.

Des avantages des conteneurs vs VM :

- comparé à une VM les ressources allouées, RAM et disques sont juste celles suffisantes au besoin du processus à faire tourner. Alors qu'il faut configurer et verrouiller des tailles en statique pour une VM.
- Un conteneur démarre beaucoup plus rapidement qu'un VM
- Les postes de développeur peuvent utiliser des conteneurs, alors que l'install de VM est plus lourde et parfois réservée à certains postes.

L'inconvénient d'un conteneur est que l'isolation est moins forte qu'une VM.

Docker existe depuis 2013.

Un conteneur Docker ne peut faire tourner qu'un seul processus (plusieurs pour les conteneurs LXC et OpenVZ).

Ainsi, dans le cas d'une stack LAMP (Linux, Apache, MySQL, PHP), nous devons créer **3 conteneurs différents**, un pour Apache, un pour MySQL et un dernier pour PHP .

Sans le système de conteneur il faut prendre de multiples précautions :

- au moment du développement en équipe, s'assurer que l'on travaille sur les postes de développement avec les même versions des logiciels et framework
- s'assurer que sur les machines de tests il y ait les mêmes versions en particulier en intégration continue
- idem lors de la mise en production en particulier en livraison continue
- il peut arriver que l'on doive développer plusieurs applications avec des versions différentes du même langage ...

L'isolation des conteneurs Docker résout ces problèmes.

Types de conteneurs : Stateless et Stateful

Il faudra choisir le type de conteneur

L'aspect immuable.

**L'immuabilité** d'un conteneur est aussi importante. Un conteneur ne doit pas stocker de données qui doivent être pérennes, car il les perdra (à moins que vous les ayez pérennisées). Mais si vous souhaitez en local mettre une base de données dans un conteneur Docker, vous devez créer un **volume** pour que celui-ci puisse stocker les données de façon pérenne.

## 2.2 Les versions de Docker

Docker Inc distribue 3 versions de Docker différentes :

- Docker Community Edition (Linux seulement) ;
- Docker Desktop (Mac ou Windows) ;
- Docker Enterprise (Linux seulement).

Nous utiliserons les versions gratuites Community pour Linux et Desktop pour les autres.

## Section 3 Installer Docker

---

Suivre le lien suivant : <https://openclassrooms.com/fr/courses/2035766-optimisez-votre-deploiement-en-creant-des-conteneurs-avec-docker/6211390-installez-docker-sur-votre-poste>

Il faudra créer un compte sur le Docker Hub et choisir la formule Personal.

Il est possible que vous deviez aussi mettre à jour la version noyau Linux WSL2 de votre machine. Laissez vous guider.

## Section 4 Un premier conteneur Docker

---

### 4.1 Lancer Docker Desktop

L'installation précédente fait que sur la machine tourne un démon docker.exe qui se comporte comme un serveur. Il va attendre des instructions de clients.

Lancer l'application Docker Desktop depuis son icône.

Si elle vous propose de lancer un getting-started, faites le.

Dans ce cas dans l'onglet Containers on voit ce container dans l'état Running, avec le port 80

Depuis un navigateur Web, taper localhost et constater qu'un serveur http tourne pour présenter Docker.

### 4.2 Exemple avec un serveur Nginx

Ouvrir une fenêtre commande. Essayez ces quelques commandes :

docker ps	montre les conteneurs en cours d'exécution
docker images	montre les conteneurs présents sur la machine

Voici d'autres commandes à utiliser plus tard :

docker pull <nom d'image>	récupère une image depuis le Docker Hub
docker run -it <nom d'image>	lance une image récupérée préalablement
docker stop <container id>	arrête l'exécution d'un container

La commande suivante va chercher un conteneur existant sur Docker Hub Nginx puis lance le serveur Nginx contenu dans ce conteneur :

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

-d pour **détacher le conteneur** du processus principal de la console. Il vous permet de continuer à utiliser la console pendant que votre conteneur tourne sur un autre processus

-p pour définir **l'utilisation de ports**. Dans notre cas, nous lui avons demandé de transférer le trafic **du port 8080 vers le port 80 du conteneur**. Ainsi, en vous rendant sur l'adresse `http://127.0.0.1:8080` , vous aurez la page par défaut de Nginx.

Dans un navigateur web utiliser l'url localhost:8080

Nous allons « rentrer » dans le conteneur Nginx :

`docker ps` pour récupérer le container id de nginx

`docker exec -ti <CONTAINER ID> bash`

Les commandes sont des commandes UNIX.

Par exemple pour trouver le fichier html affiché :

`cd /usr/share/nginx/html`

`docker rm <CONTAINER ID>`

supprimer un container

`docker system prune`  
ressources

supprime les conteneurs à l'arrêt et des

L'usage de l'appli Docker Desktop rend ces mêmes services.

Regardez là et utilisez là.



## Section 5 Créer une image Docker élémentaire

---

Pour cela il faut écrire un fichier Docker File qui décrit comment réaliser l'image.

Chaque instruction du fichier décrit une passe, une étape de la construction de l'image : une **layer**

Une première image Docker :

Utiliser un éditeur de texte, VS Code par exemple. Accepter le plugin Docker proposé au 1<sup>er</sup> emploi.

Dans un répertoire dédié, créer le fichier dockerfile avec le contenu suivant :

```
FROM alpine:3.14
```

```
CMD ["echo", "Hello World ..."]
```

alpine est une distribution d'un environnement Linux

echo est dans ce contexte une commande Linux.

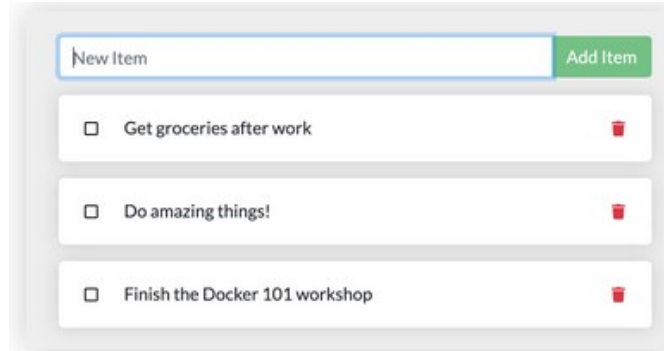
<b>docker build -t mon_image</b>	■	Pour créer l'image
docker images		montre l'existence de l'image
docker run mon_image		lance le container

## Section 6 Image avec NodeJS

---

### 6.1 But

Nous allons récupérer un exemple écrit avec Nodejs qui affiche un combo Box :



Nous allons construire une image Docker.

Une mise à jour de l'image sera faite

### 6.2 Mise en œuvre

Vérifier que le conteneur docker/getting-started est toujours Running.

Utiliser l'URL localhost dans un navigateur.

Nous allons reproduire l'exemple fourni <http://localhost/tutorial/our-application/>

Vous pouvez suivre le texte de cette page ou ce qui suit :

- Créer sur votre disque un répertoire propre à l'exemple.  
Exemple : **my-getting-started**
- Récupérer le zip proposé sur la page

#### 1. Download the ZIP.

- Dézipper dans le répertoire et parcourir les répertoires
- Avec un éditeur (VS Code) regarder le fichier package.json . C'est un fichier de configuration de NodeJs
- Créer un fichier de nom Dockerfile (sans suffixe) avec ce contenu :

```
FROM node:18-alpine
WORKDIR /app
COPY . .
RUN yarn install --production
CMD ["node", "src/index.js"]
```

- node:18-alpine est un conteneur de nodeJs utilisant la fourniture Linux Alpine

- ici yarn est une commande de NodeJs qui remplace l'outil npm
- CMD indique la commande lors du lancement du conteneur. Ici lancer le serveur NodeJs avec index.js
- dans une fenêtre commande se placer dans le répertoire app et taper la commande suivante pour générer l'image Docker

```
docker build -t my-getting-started .
```

- Lancer le conteneur par la commande :

```
docker run -dp 3000:3000 getting-started
```

- Regarder l'application à l'URL localhost:3000 et l'utiliser, créer quelques items.

### 6.3 Effectuer une modification

Nous allons remplacer le texte par défaut  
No items yet! Add one above!

Par le texte  
You have no todo items yet! Add one above!

Editer par VS Code le fichier

src/static/js/app.js

Réaliser la modification.

Ré utiliser la même commande pour construire le conteneur :

```
docker build -t my-getting-started .
```

Puis relancer le conteneur :

```
docker run -dp 3000:3000 my-getting-started
```

Vous devez constater une erreur au lancement, le port 3000 est déjà utilisé. Ceci est dû au fait que le précédent conteneur est toujours Running.

En fenêtre commande faire :

```
docker ps
```

pour la liste des conteneur running

```
docker stop <CONTAINER ID>
```

```
docker rm <CONTAINER ID>
```

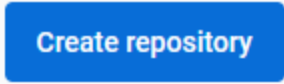
pour supprimer ce conteneur

Regarder le résultat obtenu.

## 6.4 Partager l'application sur Docker Hub

Aller sur Docker Hub et se logger. <https://hub.docker.com/>

Dans la page repositories clic sur le bouton



Create repository

Mettre le nom my-getting-started

Laisser le mode Private

Clic sur Create

Noter le nom donné à ce repository : de la forme

<nom docker>/my-getting-started

C'est ce nom qui devra être fourni lors d'un envoi par push.

Dans la fenêtre console

docker images (ou docker image ls) donne la liste des conteneurs

On peut constater que <nom docker>/my-getting-started n'est pas trouvé.

Il faut renommer le conteneur my-getting-started

```
docker tag my-getting-started YOUR-USER-NAME/my-getting-started
```

docker images montre la modification.

Se logger :

```
docker login -u <name>
```

Envoyer sur docker Hub

```
docker push <name>/my-getting-started
```

Regarder le résultat sur le dépôt Docker Hub.

## 6.5 Essayer notre conteneur

Aller sur <https://labs.play-with-docker.com/>

Se logger avec le compte Docker

Clic sur Add New Instance

Dans la console qui s'ouvre :

```
docker run -dp 3000:3000 YOUR-USER-NAME/my-getting-started
```

Ensuite un clic sur le bouton



lance notre application.

## 6.6 Réaliser une persistance de la base de données

Dans notre combo box, pour l'instant les items ajoutés sont perdus à chaque lancement du conteneur.

Chaque conteneur a son propre espace de stockage, même si plusieurs conteneurs sont lancés depuis la même image. Ceci permet une isolation des conteneurs, comportement voulu par défaut.

Si l'on souhaite avoir une persistance il faut déclarer un Volume nommé.

1) Créer un volume nommé « perBD » par la commande :)

```
docker volume create perBD
```

2) regarder son existence dans Docker Desktop

3) Arrêter le conteneur my-getting-started.

4) Affecter le volume nommé avec un nouveau conteneur par :

```
docker run -dp 3000:3000 -v perBD:/spec/persistence my-getting-started
```

5) Ajouter des items au combo box

6) Arrêter à nouveau le conteneur my-getting-started

7) Le relancer à nouveau par la même commande

```
docker run -dp 3000:3000 -v perBD:/spec/persistence my-getting-started
```

8) Constater que les items sont conservés.

9) Où se trouve stocké le volume ? Utiliser la commande :

```
docker volume inspect perBD
```

10) Il est possible de lancer un autre conteneur sur le même volume :

```
docker run -dp 4000:3000 -v perBD:/spec/persistence my-getting-started
```

regarder le résultat sur le port 4000 ...

## 6.7 Autre type de persistance avec Bind Mounts

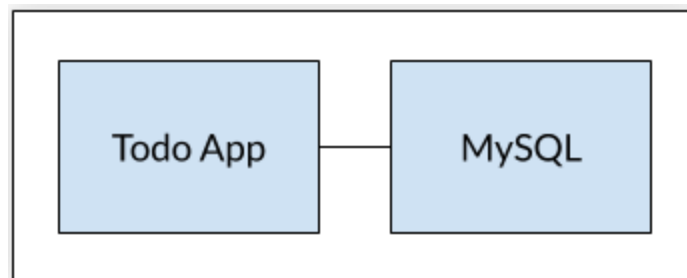
Rédaction ultérieure si utile.

## 6.8 Application multi conteneurs

<http://localhost/tutorial/multi-container-apps/>

Le plus simple avec docker est d'utiliser un conteneur par process, cas de l'appli précédente.

On se propose maintenant d'utiliser l'appli précédente avec MySQL.



MySQL nécessite un serveur DB, donc un process différent du Process serveur HTTP : il faut deux conteneurs.

Nous allons implémenter la notion de Container Networking.

La condition pour que 2 conteneurs s'échangent des données est qu'ils soient dans le même Networking.

1) Créer le Networking

Utiliser la commande :

`docker network create todo-app`

ici todo-app est le nom que nous donnons à ce Networking

`docker network list` fournit la liste des networks présents

Utiliser la commande suivante pour lancer Mysql dans un conteneurs

```
docker run -d --network todo-app --network-alias mysql -v todo-
mysql-data:/var/lib/mysql -e MYSQL_ROOT_PASSWORD=secret -e
MYSQL_DATABASE=todos mysql:8.0
```

Pour s'assurer que Mysql fonctionne nous allons l'utiliser :

Créer un client dans la console :

`docker exec -it <mysql-container-id> mysql -p`

docker pas permet d'obtenir l'id <mysql-container-id>

Réaliser ensuite quelques commandes SQL simples.

Pour que notre application utilise Mysql par le même network il faut encore un autre composant :

`docker run -it --network todo-app nicolaka/netshoot`

Dans le prompt interne à ce conteneur taper

`dig mysql`

Dans la partie ANSWER SECTION, relever l'adresse IP allouée à mysql

```
;; ANSWER SECTION:
mysql.                600      IN      A       172.18.0.2
```

Exécuter ensuite la commande pour lancer un conteneur sur l'application Node :

```
docker run -dp 3000:3000 --network todo-app -e MYSQL_HOST=mysql -e
MYSQL_USER=root -e MYSQL_PASSWORD=secret -e MYSQL_DB=todos my-
getting-started
```

Réaliser quelques modifications depuis la page Web.

Regardons dans la base de données :

IMIE

```
docker exec -it <mysql-container-id> mysql -p todos
```

Regarder le contenu de la table et constater vos modifications.



## Section 7 Utilisation de docker compose

---

### 7.1 But

Dans ce qui précède nous avons gérer manuellement le fonctionnement interconnecté de plusieurs conteneurs.

Nous allons le faire d'une autre façon ici.

### 7.2 Utilisation de compose

Avec compose, les commandes vont être lises dans un fichier YAML. Le lancement devient plus simple.

Dans notre projet app, créer le fichier docker-compose.yml au même niveau que package.json

Commenter par la ligne

services :

Nous allons décrire chaque conteneur.

Pour app la commande précédente était :

```
docker run -dp 3000:3000 \
  -w /app -v "$(pwd):/app" \
  --network todo-app \
  -e MYSQL_HOST=mysql \
  -e MYSQL_USER=root \
  -e MYSQL_PASSWORD=secret \
  -e MYSQL_DB=todos \
  node:18-alpine \
  sh -c "yarn install && yarn run dev"
```

Dans le yaml la syntaxe équivalente est :

```
services:
  app:
    image: node:18-alpine
    command: sh -c "yarn install && yarn run dev"
    ports:
      - 3000:3000
    working_dir: /app
    volumes:
      - ./:/app
    environment:
      MYSQL_HOST: mysql
      MYSQL_USER: root
      MYSQL_PASSWORD: secret
      MYSQL_DB: todos
```

Pour le conteneur Mysql, la commande précédente était :

```
docker run -d \
  --network todo-app --network-alias mysql \
  -v todo-mysql-data:/var/lib/mysql \
  -e MYSQL_ROOT_PASSWORD=secret \
  -e MYSQL_DATABASE=todos \
  mysql:8.0
```

Ceci devient dans le yaml :

```
mysql:
  image: mysql:8.0
  volumes:
    - todo-mysql-data:/var/lib/mysql
  environment:
    MYSQL_ROOT_PASSWORD: secret
    MYSQL_DATABASE: todos

volumes:
  todo-mysql-data:
```

Assurez vous qu'aucun conteneur mysql et my-getting-started tournent encore .  
Par précaution détruire ces conteneurs.

Lancer composer par la commande :

```
docker compose up -d
```

Le fichier yaml est alors utilisé  
-d indique exécution en background

Regarder ensuite le résultat avec Docker Desktop  
Utiliser la commande  
docker compose logs -f

Vérifier que le site fonctionne par localhost:3000

Si l'on souhaite arrêter les conteneurs par compose :

```
docker compose down
```

ou docker compose down --volumes                    pour supprimer aussi le volume

## Section 8 Docker : les bonnes pratiques

---

Vérifier les vulnérabilités par :

docker scout cves my-getting-started

Le résultat montre les problèmes éventuels.

Si l'on souhaite voir ces vulnérabilités, ajouter au Docker Desktop l'option Snyk

