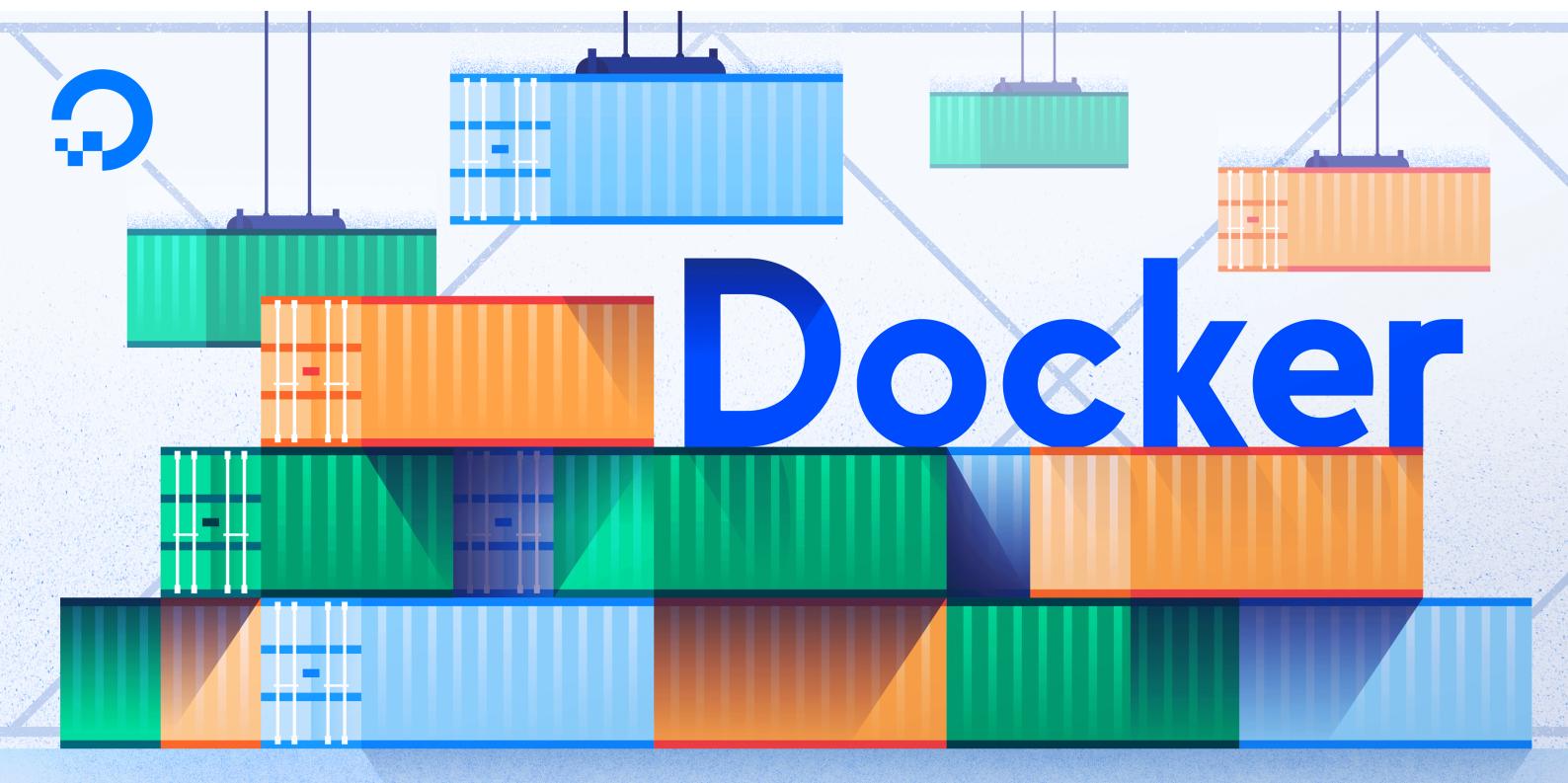


Docker - Guide pratique

Livre



Sommaire

1. Introduction	1
1.1. Préambule	1
1.2. Docker? C'est quoi ?	1
1.3. Pourquoi Docker et les conteneurs ?	6
1.4. Des environnements de développement et de production identiques	6
1.4.1. Exemple : une application NodeJs.....	7
1.5. Des environnements de développement identiques au sein d'une équipe ou d'une organisation.....	9
1.6. Des environnements conflictuels entre différents projets	9
1.7. Machines Virtuelles vs Conteneurs Docker	11
1.8. Un ordinateur dans l'ordinateur	11
1.9. Les machines virtuelles sont lourdes.....	13
1.9.1. Calcul des ressources restantes.....	14
1.9.1.1. Calcul des ressources restantes	14
1.9.1.2. Réflexion	15
1.9.1.3. Conclusion	15
1.10. Les conteneurs sont légers.....	17
2. Installation de Docker	21
2.1. Vue d'ensemble	21
2.2. Installation de Docker sur Windows	23
2.2.1. Le nouveau Terminal pour Windows	23
2.2.2. Téléchargement.....	25
2.2.3. Virtualisation d'un noyau Linux.....	26
2.2.3.1. Avec WSL	26
2.2.3.1.1. Prérequis : version du système d'exploitation	26
2.2.3.1.2. Prérequis : Configuration matérielle	27
2.2.3.1.3. Prérequis : Activer WSL.....	27
2.2.3.1.4. Mise à jour de WSL	28
2.2.3.1.5. Installation	28
2.2.3.1.6. Vérification de la version de WSL	28
2.2.3.2. Avec Hyper-V	29
2.2.4. Installation de Docker Desktop	30
3. Les images Docker	31
3.1. Introduction aux images Docker	31
3.2. Images et Conteneurs : Quelle différence ?	31
3.3. Les images Docker pré-construites.....	33
3.3.1. Préambule	33
3.3.2. Utiliser une image existante	34

3.4. Les images Docker personnalisée	37
3.4.1. Préambule	37
3.4.2. TD : Créer une image pour une application Node.js	38
3.4.2.1. Objectif	38
3.4.2.2. Préparation	38
3.4.2.3. Présentation des fichiers	38
3.4.2.4. Lancez l'application Node.js en Local	40
3.4.2.5. Création de notre propre image Docker	42
3.5. Exercices	42

1. Introduction

1.1. Préambule

Le cours que vous allez lire va aborder deux technologies qui facilitent le déploiement et le développement d'applications complexes :

- Docker
- Kubernetes

Certainement, qu'en ce moment même, vous n'avez absolument aucune idée de ce qu'est Docker et à quoi cela sert concrètement.

Docker est lié à ce que l'on appelle des **containers**.

Tout au long de ce cours, vous recevrez des commandes à tester avec leur explication.

Ainsi, vous saurez comment :

- Installer et utiliser Docker localement sur votre machine.
- Utiliser Docker dans votre projet simple comme dans les plus complexes.
- Déployez ces projets et des applications.
- Apprendre ce qu'est Kubernetes et son lien avec Docker.

Vous rencontrerez beaucoup de petits exemples pour vous plonger dans des cas pratiques et concrets.

Nous allons commencer de zéro.

Pré-requis pour suivre ce cours



- Aucune connaissance préalable de **Docker** n'est requise.
- Aucune connaissance préalable de **Kubernetes** n'est requise non plus !

1.2. Docker? C'est quoi ?



Voici une définition académique de Docker :

Docker est une technologie de conteneurisation : un outil pour créer et gérer des conteneurs.

Mais finalement, nous ne sommes pas plus avancés !

Qu'est-ce qu'un **conteneur** ? et pourquoi aurions-nous besoin, à notre niveau, de les utiliser, puisque jusqu'à présent, nous vivions très bien sans ?

Vous étudiez le développement d'applications n'est-ce pas ? Dans ce cas, vous connaissez au moins les bases d'un langage de développement comme le **Javascript**, le **PHP**, **Java** ou **C#**.

Donc, vous n'êtes pas sans savoir qu'une application est le produit d'un long processus d'écriture de lignes de codes, sur une machine de développement fonctionnant avec des librairies. L'ensemble tournant dans un environnement particulier.

Par exemple, imaginons que nous développons une application de gestion pour le football. Cette application permet d'ajouter des joueurs et leur palmarès dans des équipes et de télécharger la composition de ces équipes en format PDF.

Au lieu de programmer de zéro comment convertir nos données en PDF, nous allons utiliser des morceaux de code déjà écrits par d'autres développeurs. Ces morceaux de code sont regroupés dans ce qu'on appelle une 'librairie'. En utilisant une librairie, nous gagnons du temps et nous assurons que la fonctionnalité est déjà bien testée et fonctionnelle.

Attention toutefois, notre application est développée en PHP 7.2, une version un peu ancienne. Il faudra donc choisir une librairie qui est compatible avec cette version de PHP. Si nous ne faisons pas attention à cette compatibilité, nous risquons de rencontrer des problèmes ou des 'bugs' lors de l'utilisation de notre application. C'est un peu comme s'assurer que les pièces d'un puzzle s'emboîtent bien ensemble.

Jusqu'à maintenant, nous ne voyons pas vraiment de problèmes !

Sur notre machine de développement, nous devrons seulement installer PHP 7.2 et la librairie compatible. Idem sur le serveur de production.

Machine de développement : Windows 11

Environnement PHP 7.2

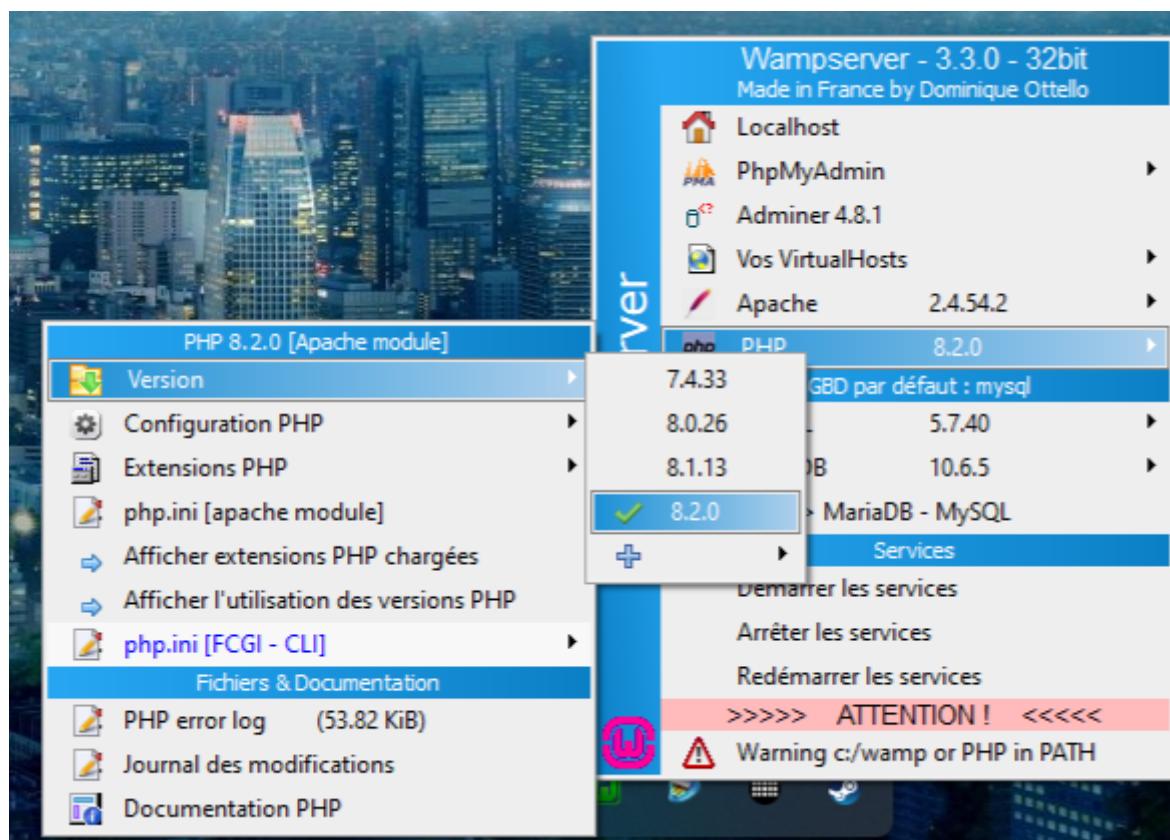
index.php equipes.php joueurs.php Fichier3.php

Librairie_export_pdf.php

Exemple : Application de gestion d'équipes de Football

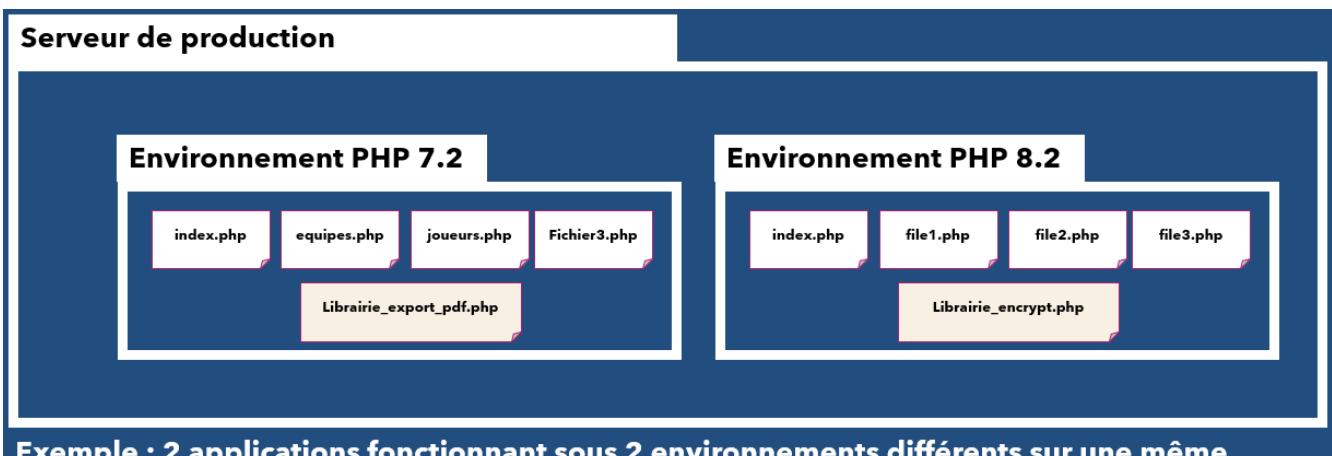
Maintenant, imaginons qu'une autre mission consistera à développer une nouvelle application sous PHP 8.2, et qui aura besoin entre autres, d'une librairie d'encryptage de mot de passe par exemple.

Sur votre machine de développement, vous utilisez sûrement **WampServer** qui permet en quelques clics de changer de version de PHP en cours :



Les choses se compliqueront durant la mise en production :

Comment faire tourner sur votre serveur les deux applications en même temps ?

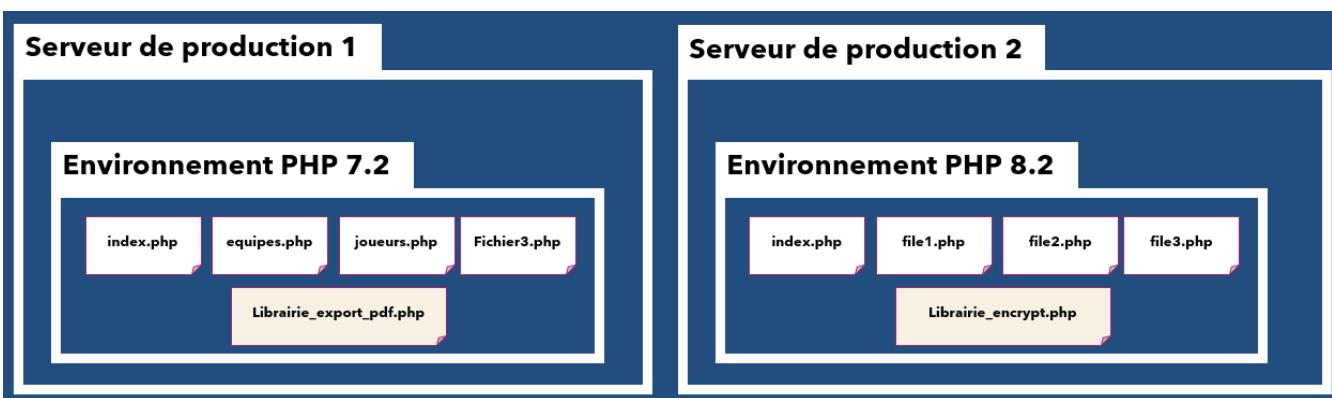


Exemple : 2 applications fonctionnant sous 2 environnements différents sur une même machine.

La première solution qui nous vient à l'esprit est aussi la plus coûteuse :

- Créer et maintenir 2 serveurs différents pour faire tourner nos applications.

On pourrait imaginer tout un tas de solutions, mais qui demanderaient de gros efforts de configuration rendant la maintenance des systèmes difficiles et sûrement instables !



Fermez les yeux, imaginez maintenant un outil miraculeux, qui, en quelques clics, vous permettrait de créer autant d'environnement que nécessaire ! indépendants les uns des autres, isolés, mais ouverts au dialogue , c'est là que **Docker** entre en scène !

Vous ne voyez toujours pas l'intérêt ?



Partons faire un pique-nique avec notre beau panier en osier !

Dans ce panier, il y a des couverts, du pain, de la nourriture, une nappe ! tout ce qui est nécessaire pour faire un pique-nique.

Chaque fois que vous prenez ce panier, vous retrouverez à chaque fois : les mêmes couverts, la même nappe, la même vaisselle. (*Surement pas la même nourriture, surtout si le panier n'a pas servi depuis longtemps, mais vous comprenez l'idée !*).

Sachez alors qu'un **conteneur**, est comme ce panier, chaque fois que vous en aurez besoin, vous pourrez le prendre partout où vous le désirez, et vous aurez la certitude d'avoir exactement les mêmes outils !

De plus un **conteneur** peut se répliquer à l'infini ou presque.



Le mot **conteneur**, évoque sûrement dans votre esprit cette image :



Et vous aurez raison !

Car un conteneur est une boîte, dans laquelle sont stockées des marchandises isolées du reste du monde. Les marchandises d'un conteneur ne sont pas mélangées avec celles d'un autre conteneur.

De plus, prenons l'exemple d'une marchandise qui nécessite d'être conservée au frais. Dans ce cas, le conteneur doit être équipé d'un système de refroidissement autonome. Pour éviter que la marchandise ne se renverse durant le transport, il est aussi essentiel de prévoir des systèmes de fixation à l'intérieur du conteneur. Grâce à ces équipements, le conteneur peut être chargé sur n'importe quel navire, stocké dans n'importe quel entrepôt ou port, tout en assurant que les marchandises sont maintenues à la bonne température et bien fixées.

Les conteneurs Docker suivent cette même philosophie. À l'intérieur d'un conteneur Docker, nous pouvons intégrer tout ce qui est nécessaire pour faire fonctionner notre application :

- Un environnement spécifique pour des applications web, comme PHP entre autres.
- Des bibliothèques dans les versions requises.

- La version spécifique de notre application.
- Tous les services essentiels, qu'il s'agisse de la gestion des e-mails, de la persistance des données, et plus encore.

Tant qu'une machine est équipée de Docker, nous pourrons y exécuter notre conteneur et utiliser notre application sans nous préoccuper des prérequis nécessaires à son fonctionnement. En effet, tout ce dont nous avons besoin se trouve déjà à l'intérieur du conteneur !

Docker n'est qu'un outil qui sert à construire des conteneurs !

A retenir



- Un même conteneur, s'il est dupliqué contiendra la même application et le même contexte d'exécution, peu importe la machine qui l'exécute, tant qu'elle contient Docker !
- Docker est utilisable sur tous les systèmes d'exploitations modernes.
- Docker simplifie la création et la gestion des conteneurs.

1.3. Pourquoi Docker et les conteneurs ?

Dans le chapitre précédent, au travers d'illustrations telles que le panier de pique-nique et le conteneur de marchandises, nous avons illustré le concept de conteneur.

Vous avez certainement compris que cette technologie facilite la création d'environnements distincts pour l'exécution d'applications.

Toutefois, Est-ce qu'utiliser des conteneurs est aussi utile qu'important ? Quels sont les avantages de l'utilisation de conteneurs ?

Pourquoi voudrions-nous des "boîtes" indépendantes et standardisées pour exécuter nos applications ?

Explorons les raisons pour lesquelles un développeur pourrait choisir d'utiliser des conteneurs dans le développement et le déploiement d'applications.

Bien que nous ayons déjà pris l'exemple d'une application de gestion d'équipe de football, approfondissons la question avec un cas d'utilisation légèrement plus technique.

1.4. Des environnements de développement et de production identiques

Lorsque nous développons une application, nous avons besoin d'un environnement de développement pour exécuter notre code et tester notre application. Ensuite, lorsque le moment est venu de déployer notre application, nous avons besoin d'un environnement de production pour exécuter notre application.

Dans un monde idéal, ces environnements **devraient être identiques**. Cependant, dans la réalité, il est difficile de garantir que les environnements de développement et de production sont

strictement identiques.

Par exemple, les développeurs peuvent utiliser des versions différentes de logiciels, des configurations différentes, des systèmes d'exploitation différents, etc.

Cela peut entraîner des problèmes de compatibilité et des bugs qui ne sont pas détectés avant le déploiement de l'application.

Les conteneurs peuvent nous aider à résoudre ce problème en nous permettant de créer des environnements de développement et de production identiques.

Nous pouvons créer un conteneur pour notre environnement de développement et un autre pour notre environnement de production.

Ces conteneurs peuvent être créés à partir de la même image de conteneur, ce qui garantit que les environnements sont identiques.

1.4.1. Exemple : une application NodeJs

Prenons l'exemple d'une application NodeJs. Vous ne connaissez pas NodeJs ? Pas de panique ! Pour résumer, NodeJs est un environnement d'exécution JavaScript côté serveur qui permet d'exécuter du code JavaScript en dehors d'un navigateur.

Car à l'origine, JavaScript était un langage de programmation utilisé uniquement dans les navigateurs web.

Aujourd'hui, NodeJs est utilisé pour créer des applications web, des applications mobiles, des applications de bureau, des outils de ligne de commande, etc.

Pour exécuter une application NodeJs, nous avons besoin d'un environnement NodeJs.

Cet environnement est composé de NodeJs, de notre code source et de toutes les dépendances de notre application.

Pour créer un environnement de développement, nous pouvons installer NodeJs sur notre ordinateur et y placer notre code source.

Le langage de programmation JavaScript est évolué rapidement et de nouvelles versions sont publiées régulièrement. Par conséquent, NodeJs est également mis à jour régulièrement pour prendre en compte les nouvelles fonctionnalités Javascript.

Voici un exemple de code source d'une application NodeJs simple:

```
1 import axios from 'axios';
2 const {data} = await axios.get('https://jsonplaceholder.typicode.com/users');
3 console.log(data);
```

Il s'agit d'une application NodeJs simple qui utilise la bibliothèque Axios pour interroger une adresse web distante, un service, qui renvoie en réponse une liste d'utilisateurs sous forme d'un tableau. On appelle ce genre de service Web une **API** (Application Programming Interface).

```
const { data } = await axios.get('https://jsonplaceholder.typicode.com/users');
```

Et ce que j'ai encadré en rouge, c'est un mot clé spécifique au langage JavaScript `await`. Utilisable de la sorte, c'est-à-dire à l'extérieur d'une fonction `async` dans un module, depuis la version **14.3** de **NodeJs**.

En d'autres mots si vous voulez exécuter ce code, il vous faudra une version de **NodeJs** supérieure ou égale à la **14.3**.

Nous pourrions avoir cette version installée sur notre machine locale, dans notre environnement de développement. Si nous souhaitons que notre application soit déployée sur un serveur distant afin qu'elle soit accessible à tous, nous devons également installer NodeJs sur ce serveur. Mais comment pouvons-nous nous assurer que la version de NodeJs installée sur notre serveur est la même que celle installée sur notre machine locale (version ≥ 14.3) ?

Si le serveur distant utilise une version plus ancienne de NodeJs, notre application ne fonctionnera pas tout simplement pas !

Nous aurons le message d'erreur suivant :

```
(node:3936) ExperimentalWarning: The ESM module loader is experimental.  
file:///app.mjs:4  
    await axios.get('https://jsonplaceholder.typicode.com/users')  
^^^^^  
  
SyntaxError: await is only valid in async function
```

Nous ne pourrons jamais avoir la certitude absolue que les environnements de développement et de production sont identiques.

JAMAIS ! C'est impossible.

Un **environnement de développement** est toujours différent d'un environnement de production. L'environnement est constitué de plusieurs composants, tels que le système d'exploitation, les versions des logiciels, les configurations, etc.

Cependant, nous pouvons essayer de rendre ces environnements aussi similaires que possible.

Et c'est là que les conteneurs entrent en jeu.

Nous pouvons créer un conteneur pour notre environnement de développement et un autre pour notre environnement de production.

Ces conteneurs peuvent être créés à partir de la même image de conteneur, ce qui garantit que les environnements sont identiques.



Cela doit être facile de **partager un environnement de développement commun** entre les membres de l'équipe !

1.5. Des environnements de développement identiques au sein d'une équipe ou d'une organisation

De la même manière que nous voulons que nos environnements de développement et de production soient identiques, nous voulons également que les environnements de développement de notre équipe ou de notre organisation soient identiques.

Cela permet de s'assurer que tous les développeurs travaillent dans le même environnement sur le même code source.

Cela permet également de s'assurer que les problèmes de compatibilité et les bugs sont détectés le plus tôt possible.

Si l'on reprend l'exemple de notre application NodeJs précédente. Imaginez que vous n'avez pas utilisé NodeJs depuis très longtemps et que vous avez une version 10 sur votre machine locale. Vous ne pourrez pas modifier le code source de l'application et l'exécuter sur votre machine sans quelques difficultés.

Encore une fois, vous vous dites que cela n'est pas un problème, vous pouvez mettre à jour NodeJs facilement. Et vous avez raison. Mais cela n'est qu'un exemple pédagogique, la réalité est bien plus complexe. Et ce n'est pas une ou deux librairies dont vous aurez à gérer les versions, mais des dizaines, voir des centaines.

Et si vous travaillez en équipe, vous voulez que chaque membre de l'équipe utilise la même version de NodeJs, la même version des librairies, la même version des outils de développement, etc. Mais, vous ne pouvez pas vous permettre de perdre du temps à mettre à jour les versions des librairies de chacun des membres de l'équipe.

Et c'est là que les conteneurs entrent en jeu ...



Les divergences entre les versions d'environnement peuvent souvent être à l'origine de conflits lorsqu'il s'agit de travailler sur différents projets.

1.6. Des environnements conflictuels entre différents projets

Lorsque nous travaillons sur plusieurs projets, nous pouvons avoir des environnements conflictuels entre eux.

Par exemple, nous pouvons avoir un projet qui utilise une version spécifique d'une librairie, tandis qu'un autre projet utilise une version différente de la même librairie.

Parfois même certaines versions récentes de librairies ne sont pas compatibles avec d'autres entraînant des erreurs qui pourraient être difficiles à traiter.

Entre deux projets, il faudra alors reparamétrer les versions des librairies, des outils de développement, etc.

Cela peut être très chronophage, source d'erreurs et absolument pas passionnant.

Et c'est là que les conteneurs entrent en jeu ...



Portables :
Les conteneurs Docker peuvent être exécutés sur n'importe quel système d'exploitation qui prend en charge la technologie Docker, ce qui facilite le transfert et le déploiement d'applications entre différents environnements.

Modulaires :
Les conteneurs Docker sont conçus pour encapsuler des parties spécifiques d'une application, ce qui les rend interchangeables et réutilisables.

Isolés :
Chaque conteneur Docker fonctionne de manière isolée, garantissant que l'application à l'intérieur du conteneur reste stable et cohérente indépendamment des autres conteneurs ou des changements dans l'environnement hôte.

Avec Docker, nous pouvons créer un conteneur pour chaque projet. Nous n'avons plus besoin d'installer les librairies, les outils de développement, etc. sur notre machine locale à chaque fois que nous travaillons sur un nouveau projet, car tout est déjà installé dans le conteneur et pas sur notre machine locale.

Changer de projet devient aussi simple que de changer de conteneur.

Durant la suite de ce cours, vous allez apprendre tout ce qu'il y a de nécessaire pour créer vos propres conteneurs et résoudre les problèmes que nous venons d'évoquer.

Vous êtes maintenant déterminé à apprendre Docker ? Alors, allons-y !



Nous voulons avoir **EXACTEMENT** le même environnement sur la machine de développement que sur celle de production !

Pourquoi ? Parce que nous voulons avoir l'assurance que l'application fonctionne **EXACTEMENT** comme durant les phases de tests !!

1.7. Machines Virtuelles vs Conteneurs Docker

Nous savons maintenant ce que sont Docker et les conteneurs et pourquoi les utiliser.

Les problèmes résolus par **Docker** et les **conteneurs** ont du sens et nous les avons vus précédemment.

Nous comprenons alors facilement pourquoi Docker est si populaire !!



Mais si vous avez aussi compris les principes de la virtualisation, vous n'êtes pas sans savoir que nous pouvons installer tous les environnements dont nous avons besoin sur une machine virtuelle et les utiliser comme nous le souhaitons :

- Nous pouvons installer un système d'exploitation sur une machine virtuelle et l'utiliser comme un ordinateur normal.
- Nous avons des terminaux indépendants de la machine hôte et exécuter les commandes que nous voulons pour installer ou configurer des logiciels.

Nous pouvons même créer des machines virtuelles à la volée et les détruire quand nous n'en avons plus besoin.

Dans ce cas, vous vous demandez peut-être pourquoi utiliser Docker alors que nous avons déjà des machines virtuelles qui peuvent faire la même chose.

C'est une question légitime et nous allons y répondre dans cette section.

1.8. Un ordinateur dans l'ordinateur



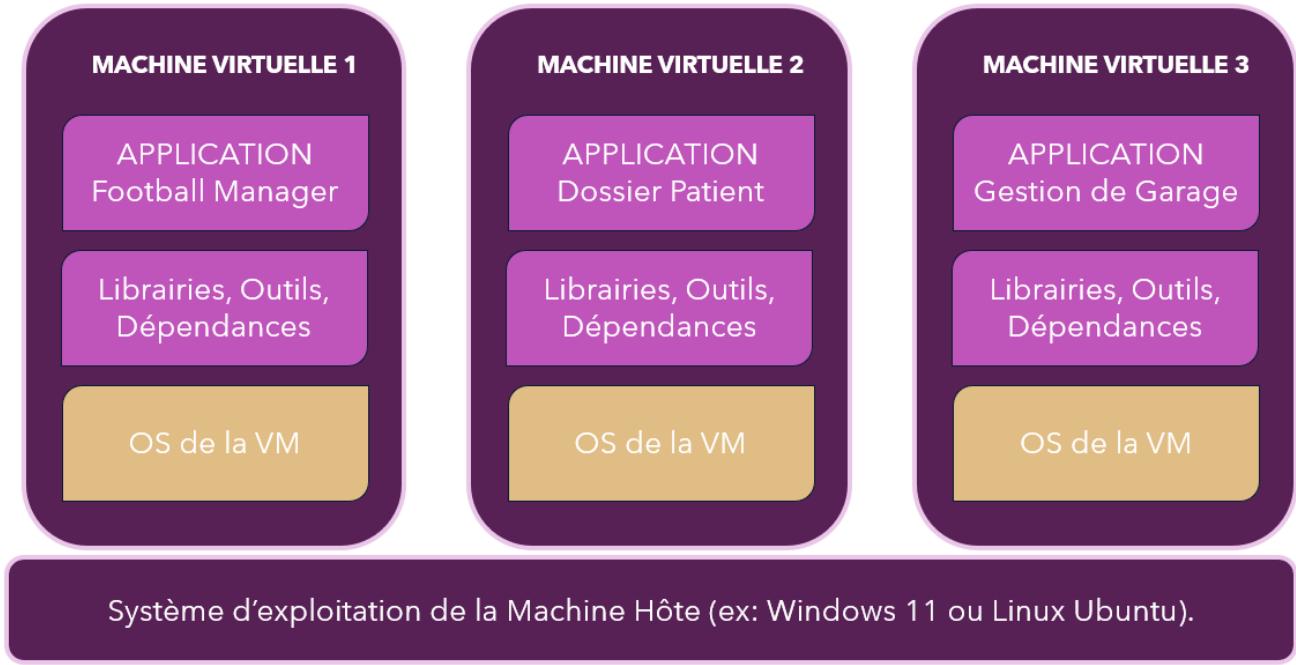
Pour virtualiser un système d'exploitation, nous avons besoin d'un logiciel nommé l'**hyperviseur** qui va créer une **machine virtuelle** et émuler le matériel pour que le système d'exploitation puisse s'exécuter.

Autrement dit : nous créons un ordinateur dans l'ordinateur.

Il existe plusieurs hyperviseurs, mais les plus connus sont **VirtualBox**, **VMWare** ou **HyperV**. Ce logiciel s'installe sur une première machine physique que l'on appelle la **machine hôte**, car c'est lui l'hôte, celui qui accueille **les machines virtuelles**.

La machine hôte possède son propre système d'exploitation et ses propres ressources matérielles (mémoire, processeur, disque dur...) Elle va donc partager ses ressources avec les **machines virtuelles** qu'elle va créer.

Donnant ainsi **aux machines virtuelles une totale indépendance**. Elles possèdent leur propre système d'exploitation et leurs propres ressources matérielles. Elles peuvent donc être utilisées comme un ordinateur normal.



Comme le représente le schéma ci-dessus, vous voyons que les machines virtuelles sont complètement autonomes les unes des autres, isolées dans leur propre coquille bien qu'elles soient toutes sur la même machine hôte.

1.9. Les machines virtuelles sont lourdes.



Les avantages que nous avons étudiés précédemment sur **Docker**, sont aussi valables pour les machines virtuelles.

- Mais alors pourquoi ne pas utiliser les machines virtuelles et devoir apprendre une nouvelle technologie ?

Pour répondre à la question, je vous propose le petit exercice de calcul suivant : calculons les ressources systèmes restant sur une machine hôte après la réaction d'une VM et menons une réflexion.

1.9.1. Calcul des ressources restantes



Voici les caractéristiques d'une machine hôte et d'une machine virtuelle installée :

Rappels

- Les ressources d'une machine virtuelle sont définies lors de son installation.
- Les ressources attribuées à une machine virtuelle ne peuvent plus être attribuées à une autre machine virtuelle ou à la machine hôte.



Retenez bien que : si vous allouez 1To de disque dur à une machine virtuelle, alors 1To de disque dur ne pourra plus être utilisé par une autre machine virtuelle ou par la machine hôte, et cela, **même si la machine virtuelle n'utilise pas tout le disque dur alloué**.

Machine	RAM	CPU	Disque dur
Machine hôte	8 Go	4 cœurs	1 To
Machine virtuelle	2 Go	2 cœurs	20 Go

1.9.1.1. Calcul des ressources restantes

Question 1 : Combien reste-t-il de ressources sur la machine hôte ?

Lors de la création d'une VM, les ressources allouées à la VM sont déduites des ressources de la machine hôte.

Pour calculer les ressources restantes, il faut soustraire les ressources de la VM aux ressources de la machine hôte.

- RAM : 8 Go - 2 Go = 6 Go
- CPU : 4 coeurs - 2 coeurs = 2 coeurs
- Disque dur : 1 To - 20 Go = 980 Go

Question 2 : Combien de machines virtuelles de ce type peut-on installer sur la machine hôte ?

Pour répondre à cette question, il suffit de diviser les ressources restantes par les ressources de la VM.

Question 3 : Puis-je installer une troisième machine virtuelle de ce type sur la machine hôte, et ne la démarrer que si les deux autres machines virtuelles sont arrêtées ?

Non, car les ressources allouées à une machine virtuelle ne peuvent pas être utilisées par une autre machine virtuelle ou par la machine hôte.

Nous sommes donc limités à deux machines virtuelles de ce type sur la machine hôte.

1.9.1.2. Réflexion

Voici quelques remarques qui vont nourrir notre réflexion :

- Sur 2 VMs, nous sommes obligés d'installer deux systèmes d'exploitation.

Par exemple : si ma machine hôte est sous Windows et que je veux installer une VM sous Windows, alors je dois installer intégralement Windows sur la nouvelle VM. De même, si je veux installer une VM sous Linux, je dois installer intégralement Linux sur la VM.

Cela prend beaucoup de place sur le disque dur de la machine hôte et beaucoup de ressources systèmes pour installer au final 2, 3 ou 4 fois les fichiers systèmes de Windows ou de Linux.

Plus il y a de VMs, plus la machine est ralentie, car ses ressources sont consommées.

1.9.1.3. Conclusion

Les machines virtuelles sont lourdes ! Mais toujours utiles et indispensables dans certains cas.

Vous ne pourrez jamais mettre par exemple Windows Server 2022 dans un conteneur, mais vous pourrez le faire sur une VM par exemple.

Voici un tableau illustrant les points positifs :

Avantages et inconvénients des VMs

POINTS POSITIFS

Environnements séparés et indépendants

Environnements spécifiques

La configuration peut être partagée et reproduite

POINTS NEGATIFS

Perte d'espace, duplication redondante

La performance est ralentie

Reproduire une VM sur une autre machine hôte: difficile et hasardeux

Notez le point négatif suivant :

"Reproduire une VM sur une autre machine hôte est parfois difficile et hasardeux".

Cela est dû au fait que les VMS installent des pilotes spécifiques à la configuration de la machine hôte.

- La carte réseau de la VM dépendra de la carte réseau de la machine hôte.
- La carte graphique de la VM dépendra de la carte graphique de la machine hôte.
- Le gestionnaire des processus qui tourne sur votre machine virtuelle dépendra de son OS bien entendu, mais aussi du processeur de la machine hôte.

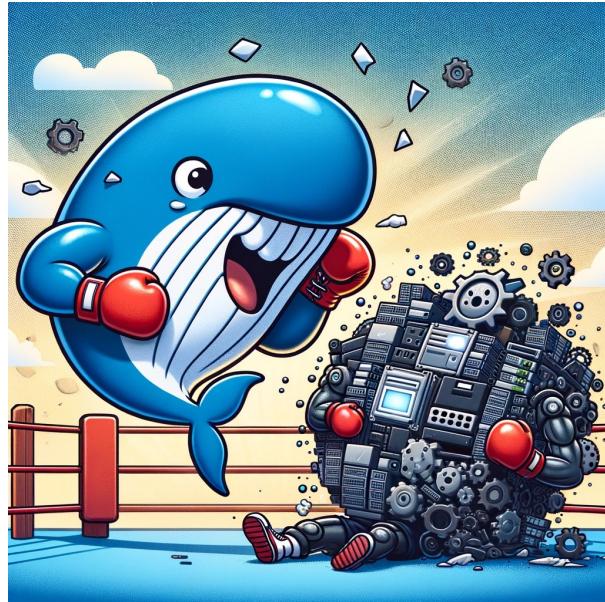
Si vous changez de machine hôte, vous devrez réinstaller les pilotes de la VM pour qu'elle fonctionne correctement.

Cela peut être très chronophage et source d'erreurs.

Alors, toujours partant de créer une VM pour seulement avoir un environnement de développement en PYTHON ?

Je crois que vous avez compris l'idée !

Nous allons donc appeler sur le ring le prochain challenger : Docker !



1.10. Les conteneurs sont légers.

Sur une machine hôte sur laquelle nous voulons installer des conteneurs, il nous faudra toujours un système d'exploitation : Windows, Linux ou MacOs.

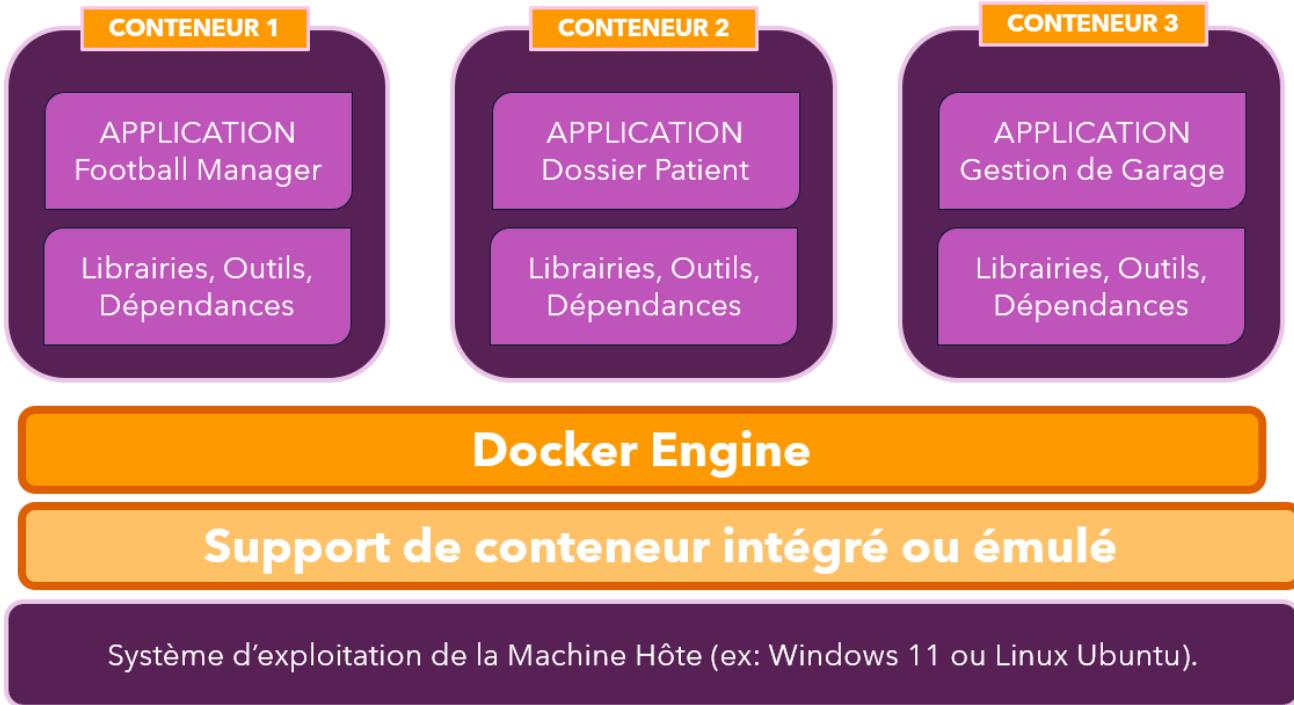
Mais nous n'installerons pas un ensemble de "machines dans la machine" via un hyperviseur.

À la place, nous allons avoir besoin d'un support de conteneur intégré (ou ***built-in container support*** en anglais) au système d'exploitation de la machine hôte ou une émulation de ce support.

Et c'est là que Docker entre en jeu ! Lors de son installation, Docker fournit un tout **petit outil et léger** nommé : "**Docker Engine**".

Cet outil va nous permettre de créer des conteneurs et de les exécuter sur notre machine hôte !!

Voyons un schéma qui illustre la place des conteneurs dans l'architecture de votre ordinateur :



Et regardez les conteneurs :

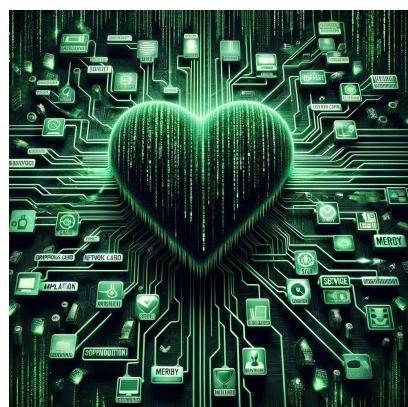
Ils sont beaucoup moins lourd que les VMs car il n'y a pas de système d'exploitation.

Nos applications seront donc plus rapides à démarrer et à exécuter. Sur le conteneur 1, il y a seulement notre application de Football ainsi que les bibliothèques nécessaires à son exécution. Rien de plus !!

Vraiment? bon ... j'ai menti ... il y a bien un système d'exploitation dans le conteneur, mais vraiment très léger !!

La machine hôte va seulement partager **son noyau** avec les conteneurs MAIS pas l'intégralité de son système d'exploitation. C'est pour cela que le conteneur doit embarquer des petits bouts de système d'exploitation qui lui sont nécessaires pour fonctionner et rien de plus.

J'imagine que vous avez bloqué sur le mot "**noyau**" ?



Le noyau est un terme que l'on rencontre souvent en informatique, son nom est "kernel" en anglais. C'est le cœur du système d'exploitation. C'est la partie qui permet au monde matériel de votre machine : carte graphique, carte réseau, processeur, mémoire, disque dur, etc. de communiquer

avec le monde logiciel : les applications, les bibliothèques, les services, etc.

Le noyau est donc la partie la plus importante du système d'exploitation.

A contrario des VMs, les conteneurs n'ont donc pas à se soucier de la compatibilité avec le matériel de la machine hôte, car ils partagent le même noyau.

Ainsi, les conteneurs peuvent se partager d'une machine à une autre, et sont beaucoup plus légers et rapide à mettre en place !

Un autre point non négligeable : La gestion des ressources systèmes est beaucoup plus fine avec les conteneurs qu'avec les VMs.

Un conteneur ne consomme que ce qu'il a besoin, et pas plus ! Si pendant 10 min, il a besoin de 4 Giga de Ram ... il va les prendre, mais les rendra aussitôt après !

L'usage de l'espace disque est dynamique : il augmente ou diminue en fonction des besoins du conteneur.

Même principe avec les coeurs du processeur.

Que dire de plus

Fin du match !

Docker a gagné sur le Ring : "Déploiement d'environnement spécifique pour vos applications" mais n'enterrez pas les VMs trop vite, elles ont encore de beaux jours devant elles !



2. Installation de Docker

2.1. Vue d'ensemble

Nous avons maintenant une bonne idée de ce que sont Docker et les Conteneurs. Nous allons maintenant passer à l'installation de Docker sur notre machine.

Les étapes d'installation de Docker dépendent du système d'exploitation que vous utilisez.



Allez sur le site officiel de Docker pour télécharger la version de Docker qui correspond à votre système d'exploitation.

- En cliquant sur le lien suivant : <http://www.docker.com>

A screenshot of the Docker website homepage. The header features the Docker logo, navigation links for Products, Developers, Pricing, Support, Blog, and Company, and buttons for Sign In and Get Started. A search bar is also present. The main content area has a dark blue gradient background. It features the text "buildcloud" with a diamond icon, "Docker Builds: Now Lightning Fast", and "Announcing Docker Build Cloud general availability". A pink button labeled "Discover Docker Build Cloud" is visible. At the bottom, there's a section titled "What is Docker?" and a large call-to-action button with the text "Accelerate how you build, share and run applications". A small tooltip message "Hey! What brings you here to..." is shown near the bottom right.

Puis cliquez sur le menu "*Developpers*" et sélectionnez "*Documentation*".

The screenshot shows the top navigation bar of the buildcloud website. The 'Developers' menu item is highlighted with a red box. A dropdown menu is open under 'Developers', containing four items: 'Documentation' (also highlighted with a red box), 'Getting Started', 'Resources', and 'Trainings'. The background of the page features the 'buildcloud' logo.

Ensuite, cliquez sur "Get Docker" :

The screenshot shows the 'Get Docker' section of the buildcloud website. It features a purple play button icon followed by the text 'Get Docker'. Below this, a subtext reads 'Learn how to install Docker for Mac, Windows, or Linux and explore our developer tools.' A red box highlights the 'Get Docker' button, and a red arrow points to it.

Et sur la page, vous trouverez les instructions pour installer Docker sur votre système d'exploitation.

Comme ici, nous voyons le lien pour installer Docker sur Windows :

The screenshot shows the 'Docker Desktop for Windows' section of the buildcloud website. It includes the Docker logo, the text 'Docker Desktop for Mac', a description 'A native application using the macOS sandbox security model that delivers all Docker tools to your Mac.', the Docker logo again, the text 'Docker Desktop for Windows', a description 'A native Windows application that delivers all Docker tools to your Windows computer.', and the Docker logo once more. A red box highlights the 'Docker Desktop for Windows' section, and a red arrow points to it.

Sur la page, vous trouverez les prérequis pour installer Docker sur Windows, Mac ou Linux.

Sur votre Système d'exploitation Windows ou Mac, vous pouvez installer **Docker Desktop** ou **Docker Toolbox**.

Docker Desktop et **Docker Toolbox** sont des outils qui donnent vie à Docker sur les systèmes d'exploitation Windows et Mac.

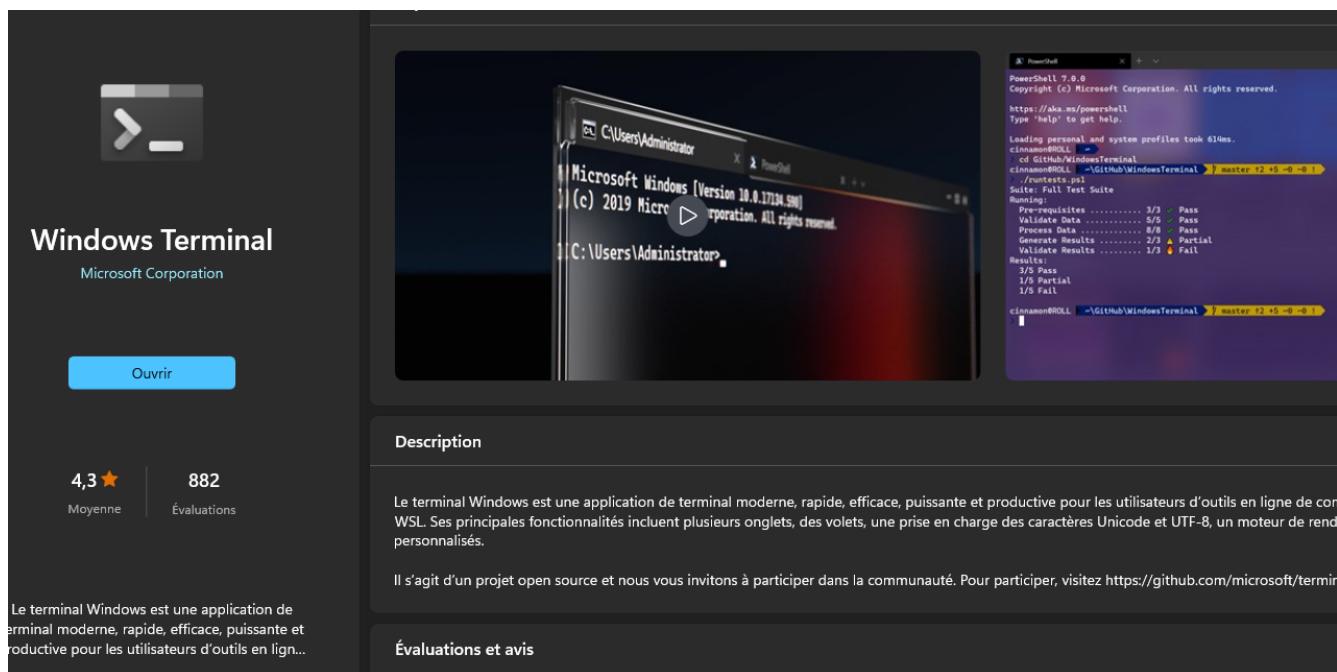
Si vous avez une machine sous Linux, vous pouvez installer **Docker Engine** directement sans avoir besoin de **Docker Desktop** ou **Docker Toolbox**. Car les **OS Linux** prennent en charge nativement les conteneurs et la technologie que Docker utilise.

2.2. Installation de Docker sur Windows

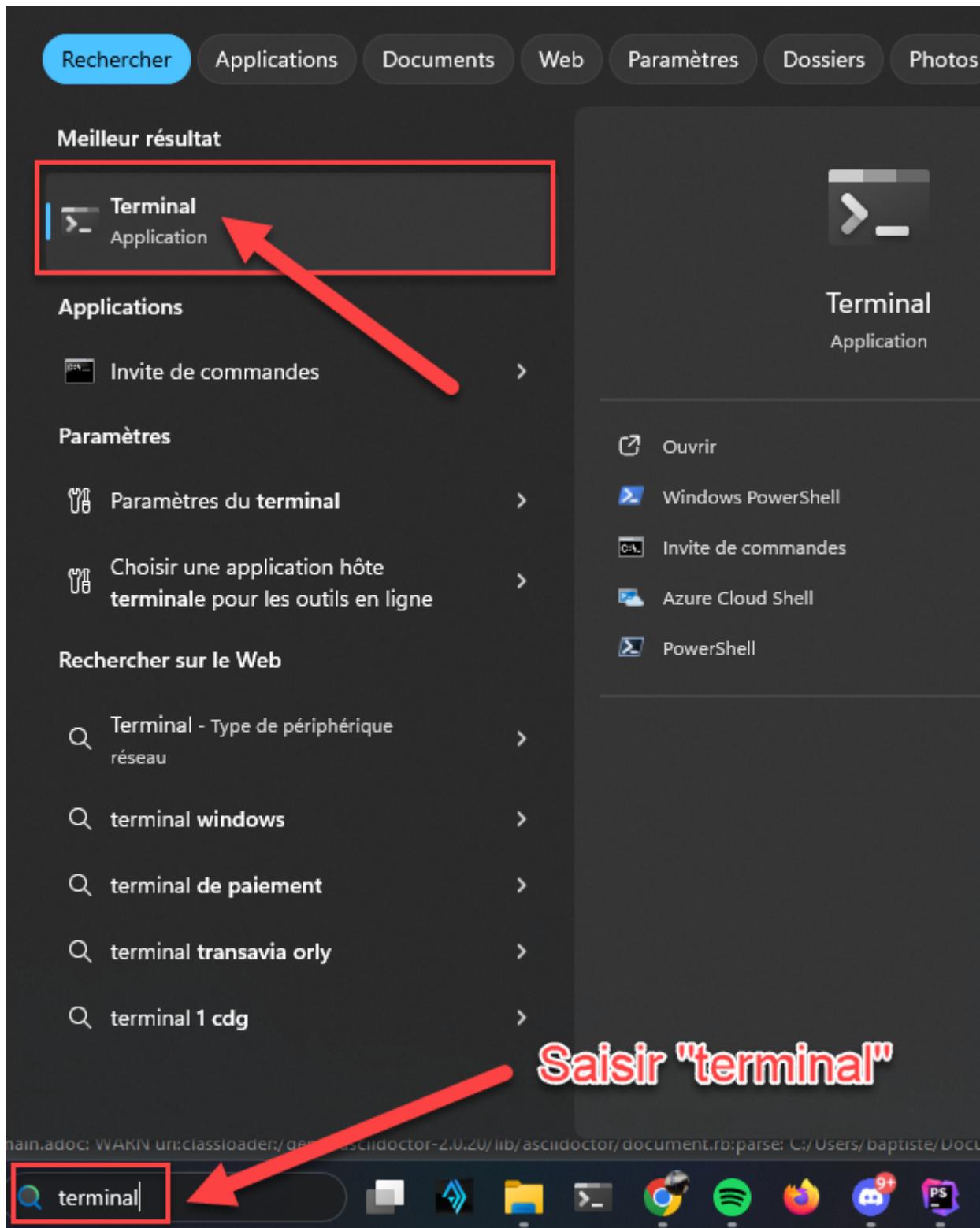
2.2.1. Le nouveau Terminal pour Windows

Durant tout ce cours, nous allons utiliser de nombreuses lignes de commande. Pour cela, nous vous recommandons d'utiliser le nouveau terminal de Windows. Qui est un outil très puissant, personnalisable, qui supporte les onglets, les thèmes, les raccourcis clavier, et bien plus encore.

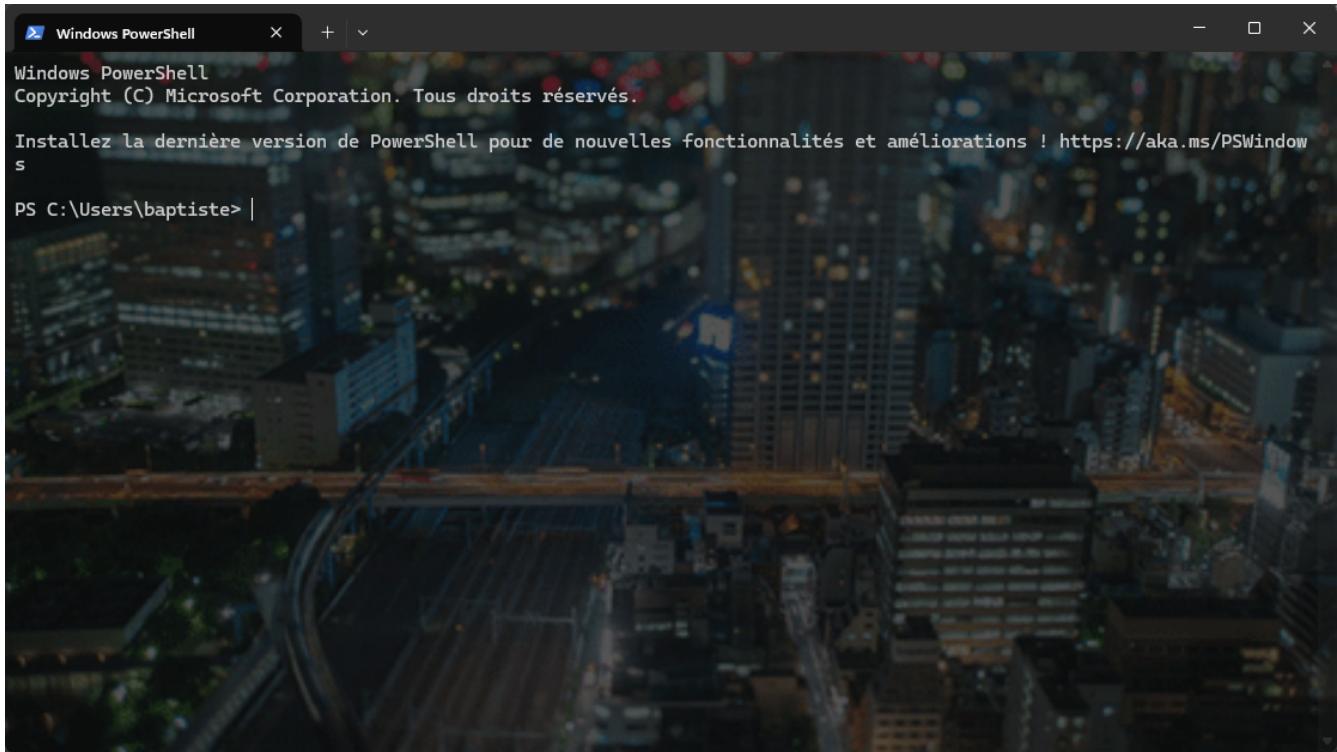
Pour installer le nouveau terminal de Windows, vous pouvez le télécharger depuis le **Microsoft Store**.



Une fois installé, vous pouvez l'ouvrir en tapant **Windows Terminal** dans la barre de recherche de Windows.



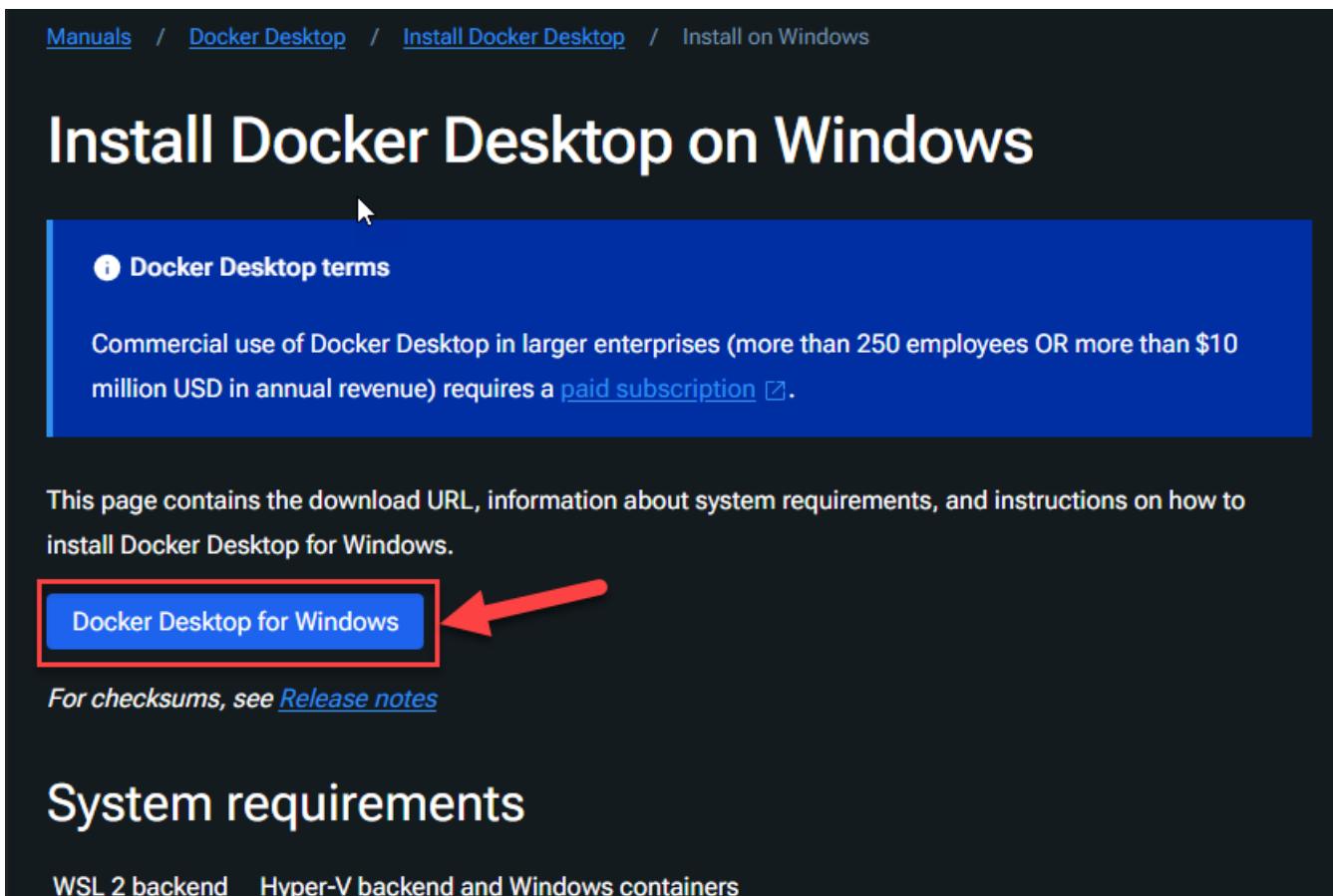
Vous aurez alors accès à un nouveau terminal moderne personnalisable comme le mien :



A screenshot of a Windows PowerShell window. The title bar says "Windows PowerShell". The content area shows a command prompt with the path "PS C:\Users\baptiste> |". The background of the window is a blurred image of a city skyline at night.

2.2.2. Téléchargement

Téléchargez Docker Desktop pour Windows



Manuals / Docker Desktop / Install Docker Desktop / Install on Windows

Install Docker Desktop on Windows

Docker Desktop terms

Commercial use of Docker Desktop in larger enterprises (more than 250 employees OR more than \$10 million USD in annual revenue) requires a [paid subscription](#).

This page contains the download URL, information about system requirements, and instructions on how to install Docker Desktop for Windows.

Docker Desktop for Windows 

For checksums, see [Release notes](#)

System requirements

WSL 2 backend Hyper-V backend and Windows containers



Lisez la suite de ce document avant d'installer Docker Desktop.

2.2.3. Virtualisation d'un noyau Linux

Comme nous l'avons vu, Docker utilise un noyau Linux pour fonctionner. Dans le cas de Windows, Docker Desktop utilise une machine virtuelle pour exécuter un noyau Linux. Il existe deux options pour exécuter Docker Desktop sur Windows :

- **WSL 2** (Windows Subsystem for Linux) : C'est une fonctionnalité de Windows 10 qui permet d'exécuter un noyau Linux sur Windows.
- **Hyper-V** : C'est une technologie de virtualisation de Windows.

Dois-je utiliser WSL 2 ou Hyper-V ?

Docker Desktop utilise **WSL 2** par défaut. Si vous avez déjà installé **WSL 2**, **Docker Desktop** l'utilisera. Toutefois, il est possible de basculer entre **WSL 2** et **Hyper-V**. Il n'y a pas de différences majeures entre les deux.

 **WSL 2** est recommandé, car il est plus rapide et plus léger qu'`Hyper-V`. Mais les deux possèdent des avantages et des inconvénients.

Par exemple, les conteneurs et les images créés avec **Docker Desktop** sont partagés entre tous les utilisateurs de la machine où **Docker Desktop** est installé. Cela est possible, parce que tous les comptes utilisateurs de la machine ont accès à la même machine virtuelle de **Docker Desktop**. Cependant, notez qu'il n'est pas possible de partager des **conteneurs** et des **images** entre plusieurs comptes utilisateurs quand vous utilisez **Docker Desktop** avec **WSL 2**.

2.2.3.1. Avec WSL

2.2.3.1.1. Prérequis : version du système d'exploitation

Il faut une machine avec :

- **Windows 11 64-bit:** **Home** ou **Pro** version 21H2 or higher, or Enterprise or Education version 21H2 or higher.
- **Windows 10 64-bit:**
 - Il est recommandé **Home or Pro 22H2 (build 19045) or higher**, or **Enterprise or Education 22H2 (build 19045) or higher**.

Si votre version de **Windows** n'est pas compatible, vous pouvez utiliser **Docker Toolbox**.

Pour vérifier la version de votre système d'exploitation, tapez **winver** dans la barre de recherche de Windows.



2.2.3.1.2. Prérequis : Configuration matérielle

Il faut que votre machine ait :

- Un processeur 64-bit avec prise en charge de la technologie SLAT (Second Level Address Translation).
- Au moins 4 Go de RAM.
- La virtualisation activée dans le BIOS/UEFI.

2.2.3.1.3. Prérequis : Activer WSL

WSL est une fonctionnalité de Windows qui permet de d'installer une distribution Linux sur Windows et d'utiliser des applications, des utilitaires et des outils de ligne de commande Bash directement sous Windows, sans modification et sans le surcoût d'une machine virtuelle traditionnelle ou d'une configuration en double démarrage.

Vérifiez si `wsl` est installé en tapant la commande suivante :

```
wsl --version
```

```
PS C:\Users\baptiste> wsl -v
Version WSL : 2.1.5.0
Version du noyau : 5.15.146.1-2
Version WSLg : 1.0.60
```

- Il faut que WSL soit à la version 1.1.3.0 ou supérieure pour pouvoir utiliser Docker Desktop.



Si vous obtenez une erreur, cela signifie que **WSL** n'est pas installé sur votre machine.

2.2.3.1.4. Mise à jour de WSL

Si WSL est déjà installé, vous pouvez le mettre à jour en tapant la commande suivante :

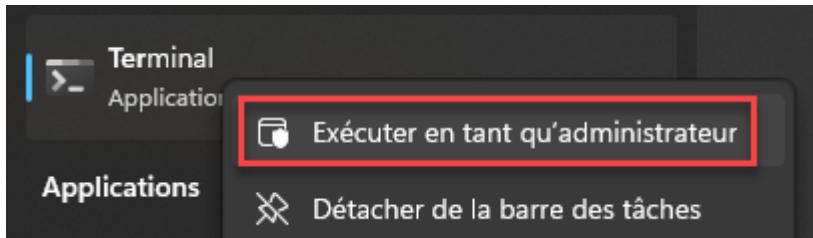
```
wsl --update
```

2.2.3.1.5. Installation

Vous pouvez désormais installer tout le nécessaire pour exécuter **WSL** avec une seule commande. Ouvrez **PowerShell** ou l'invite de commande Windows` en mode **administrateur**.

Ou bien, utilisez le nouveau terminal de Windows.

Pour cela, faites un clic droit sur l'application "**Terminal**" et sélectionnez "**Exécuter en tant qu'administrateur**". Ensuite, tapez la commande wsl --install et redémarrez votre machine. Par défaut, le nouveau terminal de Windows lance un **PowerShell**.



Saisissez la commande suivante :

```
wsl --install
```

Cette commande va installer les outils nécessaires pour lancer **WSL** sur votre machine ainsi qu'une distribution '**Ubuntu**' par défaut.

Vous pouvez changer de distribution en utilisant la commande suivante :

```
wsl --install -d <distribution>
```

En remplaçant **<distribution>** par le nom de la distribution que vous souhaitez installer.

Pour plus d'informations, vous pouvez consulter la documentation officielle de Microsoft sur WSL :

2.2.3.1.6. Vérification de la version de WSL

Sur Windows nous avons deux versions de **WSL** : WSL 1 et WSL 2. DOCKER Desktop a besoin de **WSL 2**.

Pour vérifier la version de **WSL** utilisée par votre machine, tapez la commande suivante :

```
wsl -l -v
```

Vous obtiendrez une liste des distributions installées sur votre machine avec leur version.

NAME	STATE	VERSION
* Ubuntu	Stopped	2
docker-desktop	Stopped	2
docker-desktop-data	Stopped	2

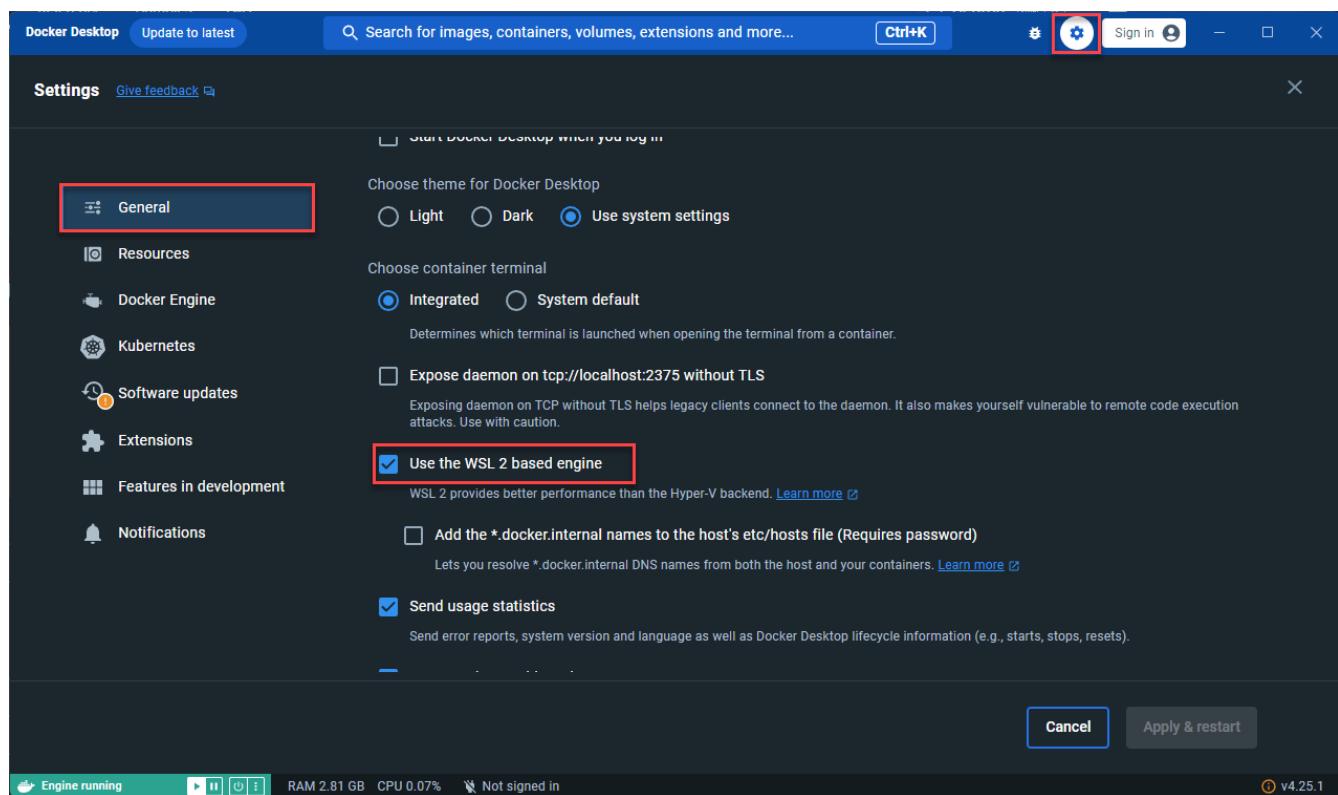
Tapez la commande suivante pour définir **WSL 2** comme version par défaut de **WSL** :

```
wsl --set-default-version 2
```

Il faut également vérifier que Docker Desktop est configuré pour utiliser **WSL 2** comme backend.

Ouvrez Docker Desktop et allez dans les paramètres en cliquant sur le petit engrenage en haut à droite de la fenêtre.

Dans l'onglet "**General**", assurez-vous que "**Use the WSL 2 based engine**" est coché.



Votre machine est maintenant prête à utiliser Docker Desktop !

2.2.3.2. Avec Hyper-V

2.2.4. Installation de Docker Desktop

Après s'être assuré d'avoir tous les prérequis nécessaires, vous pouvez installer Docker Desktop en double-cliquant sur le fichier d'installation pour lancer l'installation.

3. Les images Docker

3.1. Introduction aux images Docker

Dans ce chapitre, nous allons traiter deux concepts fondamentaux de Docker : les images et les conteneurs. Il est essentiel de connaître et de comprendre ces deux concepts pour utiliser Docker de manière efficace.

Dans la présentation de Docker, nous avons brièvement évoqué le concept de conteneurs. À présent, nous allons explorer la notion d'image Docker pour saisir le lien qui les unit aux conteneurs.

Nous allons apprendre comment utiliser des images existantes, ainsi que créer nos propres images personnalisées.

Plongeons maintenant dans ce nouveau concept pour travailler pleinement avec Docker.

3.2. Images et Conteneurs : Quelle différence ?

Comme mentionné précédemment, lorsque nous utilisons **Docker**, nous ne disposons pas seulement de conteneurs, mais aussi d'images.

Quelle est la différence entre ces deux concepts et pourquoi avons-nous besoin des deux ?

Nous savons que les conteneurs, en fin de compte, sont de petits paquets qui contiennent tout ce dont nous avons besoin pour exécuter une application : l'application elle-même, ses dépendances, ses bibliothèques, ses variables d'environnement, ses serveurs, etc. En d'autres termes, c'est l'environnement complet nécessaire pour exécuter l'application.



Un conteneur est donc un **processus**, car c'est finalement ce que nous exécutons sur notre machine.

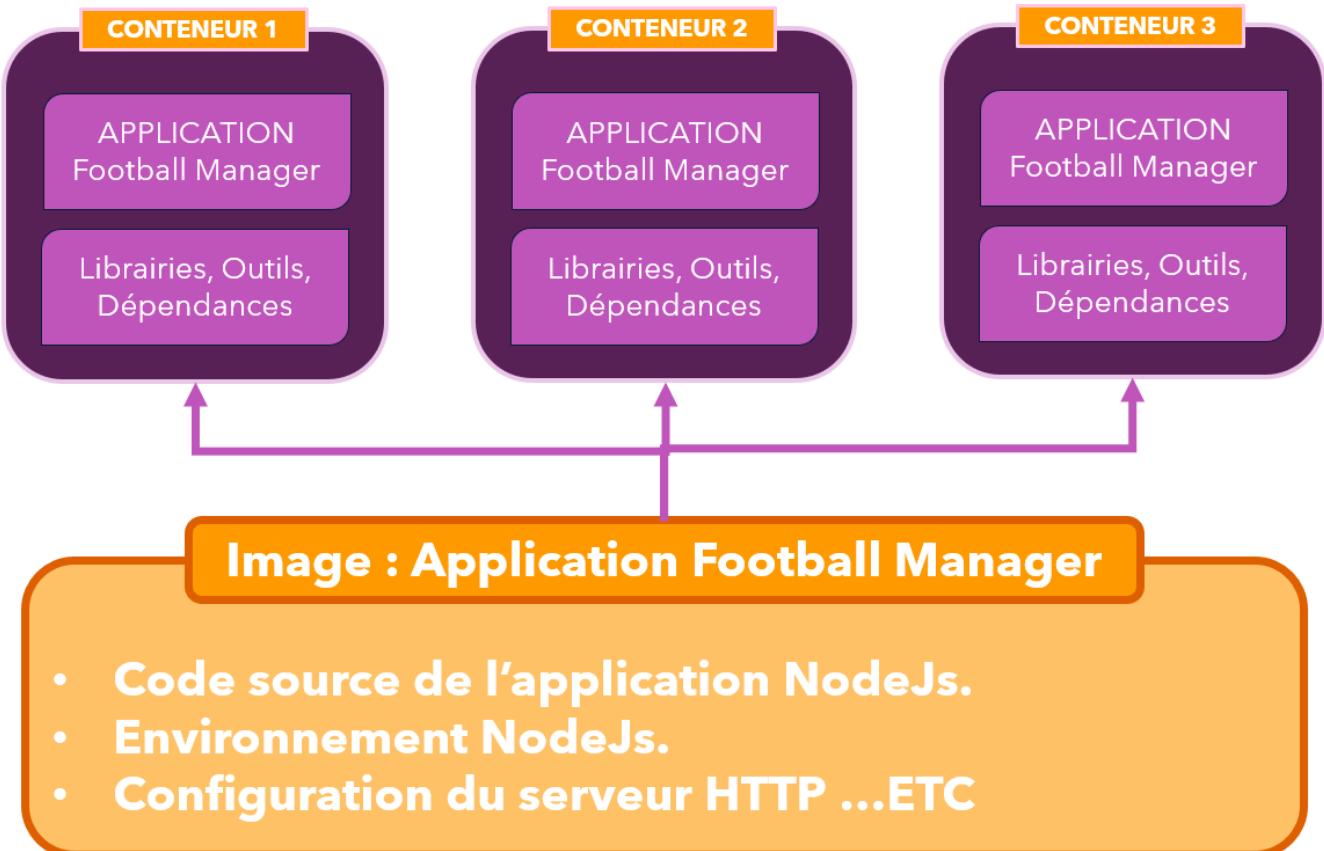
D'autre part, une image est un fichier contenant tout ce dont nous avons besoin pour créer un conteneur. Une image est un modèle, une sorte de gabarit qui nous permet de créer un conteneur. Elle contient le code source et les outils nécessaires pour exécuter une application.

Le rôle du conteneur est de lancer et d'exécuter l'application.



À partir d'une seule image, nous pouvons créer plusieurs **conteneurs** qui exécutent la même application dans le même environnement.

Prenons l'exemple d'une application web écrite en **Node.js**. Nous la définissons une seule fois dans une **image**, puis nous pouvons exécuter cette application plusieurs fois dans des conteneurs différents, sur différentes machines, sur différents serveurs.

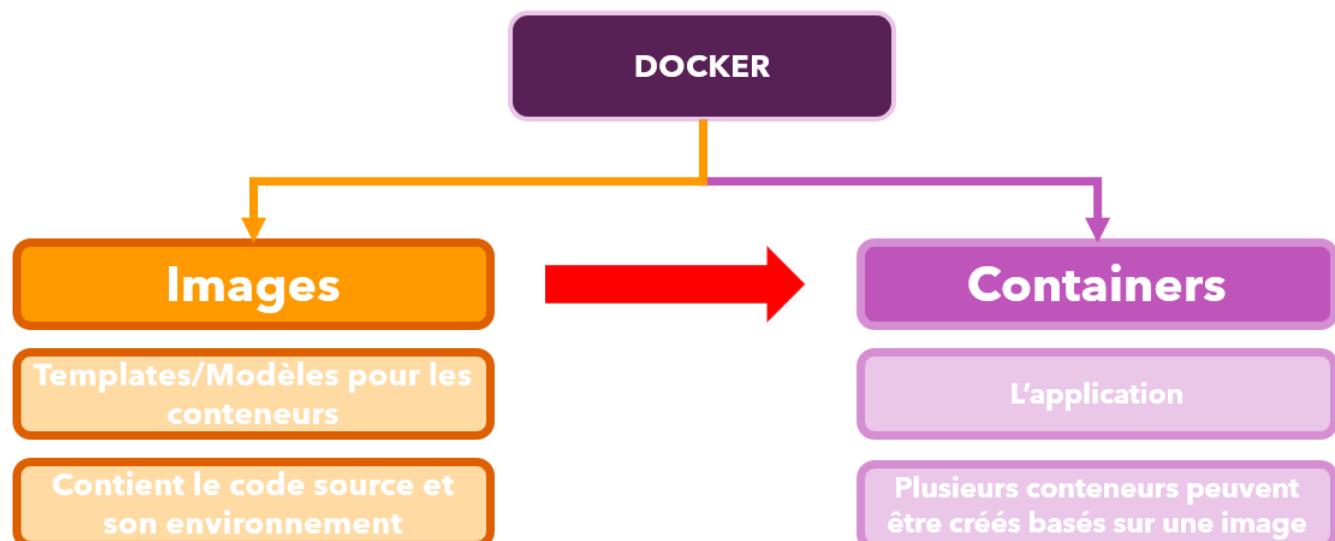


Cette image est un package partageable contenant toutes les instructions d'installation et de configuration de l'application. Le conteneur est une instance de cette image qui exécute l'ensemble des instructions.



Nous lançons des conteneurs qui sont basés sur des images. **C'est là le concept fondamental de Docker.**

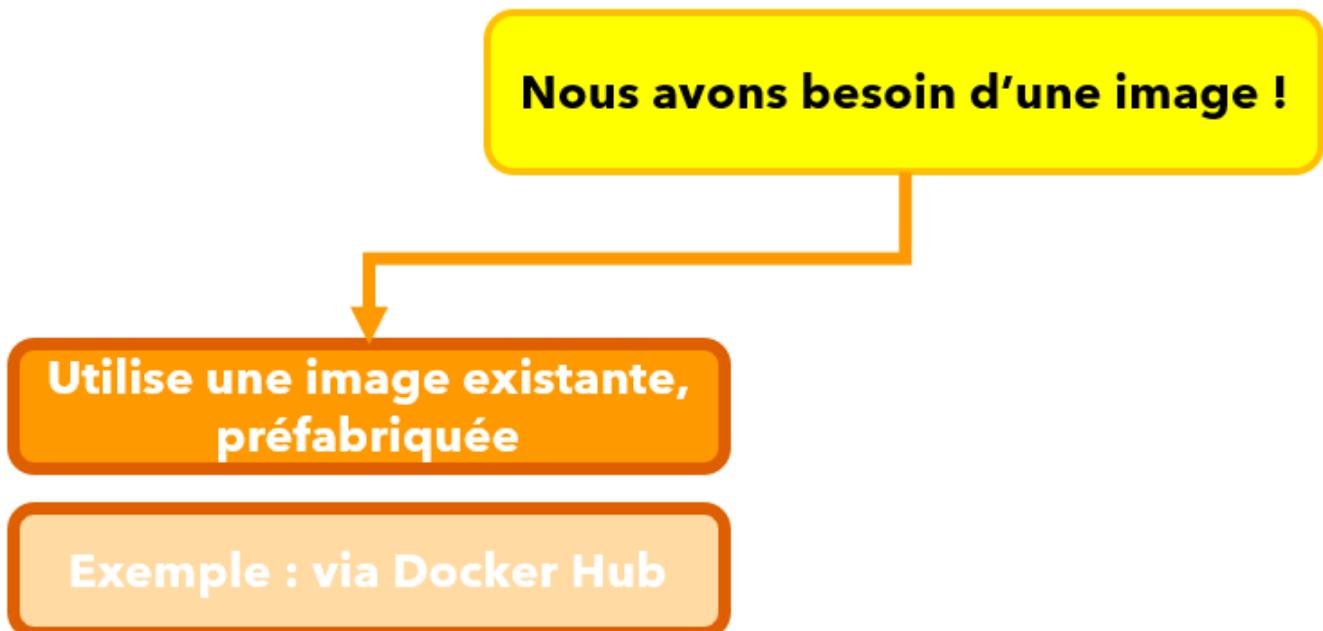
Cela deviendra encore plus clair lorsque nous commencerons à manipuler les images et les conteneurs.



3.3. Les images Docker pré-construites

3.3.1. Préambule

Il y a deux façons de créer ou d'obtenir des images Docker : Nous en étudierons une dans ce sous-titre, et l'autre dans le sous-titre suivant.



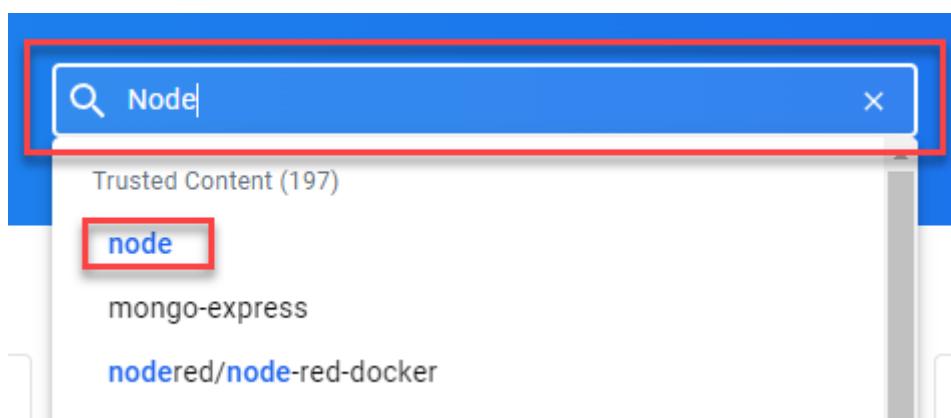
Utiliser des conteneurs existants : Ceux-ci sont créés par la communauté, nos collègues ou officiellement par les éditeurs de logiciels.

Il existe un grand nombre d'images Docker disponibles sur le Docker Hub, le registre public de Docker. Vous pouvez y trouver des images pour des applications populaires telles que MySQL, Redis, Node.js, Python, etc.

[Lien vers le Docker Hub](#)

Vous n'avez pas besoin de vous enregistrer ou de vous authentifier sur le site pour accéder aux images.

Par exemple, dans la barre de recherche, vous pouvez effectuer une recherche pour trouver l'image officielle de Node.js qui pourra être utilisée pour construire un conteneur avec Node.js.



Nous utiliserons beaucoup d'images officielles dans ce cours, mais aussi généralement dans notre travail quotidien avec Docker.

Voici la réponse du moteur de recherche de Docker Hub :

The screenshot shows the Docker Hub search results for the 'node' image. At the top, there's a search bar with the text 'docker pull node' and a 'Copy' button. Below it, the search results are displayed under the 'Tags' tab. The first result is 'lts-alpine3.19', which was last pushed 5 days ago by 'dojank'. It has four variants listed: '73753e08a875' (linux/amd64), '2a984bb09202' (linux/arm/v6), and '84801715e91d' (linux/arm/v7). Each variant includes OS/ARCH, Vulnerabilities status, and Compressed Size (45.73 MB, 44.17 MB, and 43.41 MB respectively).

Pour installer l'image, on vous donne une commande :

```
docker pull node
```

Cette commande télécharge l'image sur votre machine hôte.

Vous pouvez sélectionner une image particulière contenant une version spécifique du service ou du système Linux de base. Pour ce faire, consultez l'onglet **Tag** et récupérez le nom de tag correspondant à la version souhaitée.

Par exemple, si nous voulons utiliser **Node.js** basé sur une distribution **Alpine 3.19** :

```
docker pull node:lts-alpine3.19
```

3.3.2. Utiliser une image existante

Assurez-vous d'avoir Docker Engine de démarré sur votre machine. (Ouvrez l'application Docker Desktop et vérifiez que le statut est "Engine Running" ou "Ressource Saver mode")



Ouvrez un terminal sur votre machine hôte **et** exécutez la commande suivante pour télécharger l'image officielle de NodeJs et monter un conteneur :

```
docker container run node
```



Les commandes `docker container run` et `docker run` ont le même effet. Cependant,

depuis la **version 1.13** de Docker, il est recommandé d'utiliser `docker container run`.

En effet, l'ensemble des commandes et sous-commandes ont été réorganisées pour suivre une structure de ce type : `docker <objet> <commande> <options>`.

Cette nouvelle structure offre une meilleure lisibilité de la commande et permet de connaître son champ d'action.

Ainsi, en tapant une commande de ce type : `docker container <commande>`, nous savons que nous allons manipuler des conteneurs.

Tandis que `docker image <commande>` concerne la manipulation d'une image.

Si vous n'avez pas exécuté la commande `docker pull node` précédemment, vous verrez alors apparaître une erreur signalant que l'image `node` n'a pas pu être trouvé localement. Ainsi, elle sera automatiquement téléchargé depuis le terminal sur les serveurs du **Docker Hub**.

PS C:\Users\baptiste> docker run node
Unable to find image 'node:latest' locally
Latest: Pulling from library/node
71215d55680c: Downloading [=====] 14.2MB/49.55MB
3cb8f9c23302: Downloading [=====] 3.435MB/24.05MB
5f899db30843: Downloading [==>] 2.681MB/64.14MB
567db630df8d: Waiting
f4ac4e9f5ffb: Waiting
375735fc当地 7a: Waiting
c12db77023cd: Waiting
ac50344c1606: Waiting

Message d'erreur indiquant que l'image "node" n'a pas été trouvé localement.
Téléchargement de l'image

Maintenant que l'image est présente sur notre machine, la même commande procède à la création d'un conteneur basé sur cette image et le lance. Cependant, sur le terminal, l'action semble se terminer sans que rien se produise.

PS C:\Users\baptiste> docker run node
Unable to find image 'node:latest' locally
Latest: Pulling from library/node
71215d55680c: Pull complete
3cb8f9c23302: Pull complete
5f899db30843: Pull complete
567db630df8d: Pull complete
f4ac4e9f5ffb: Pull complete
375735fc当地 7a: Pull complete
c12db77023cd: Pull complete
ac50344c1606: Pull complete
Digest: sha256:b9ccc4aca32eebf124e0ca0fd573dacffba2b9236987a1d4d2625ce3c162ecc8
Status: Downloaded newer image for node:latest
PS C:\Users\baptiste> |

Installation terminée mais il ne se passe rien ?

Pourquoi ?



Il est essentiel de noter que par **défaut**, un conteneur est isolé de son environnement immédiat. Cependant, il est possible d'interagir avec lui via un

shell interactif. Lorsqu'il est créé, le conteneur est initialement lancé **en mode détaché**, ce qui signifie qu'il s'exécute en arrière-plan sans offrir de terminal pour interagir directement avec lui.

Donc, même si sur le terminal, il ne semble rien se passer, le conteneur a bien été créé.

Vérifions cela en exécutant la commande suivante :

```
docker ps -a
```

```
PS C:\Users\baptiste> docker ps -a
CONTAINER ID IMAGE COMMAND CREATED STATUS PORTS NAMES
3aaae58260be node "docker-entrypoint.s..." 4 minutes ago Exited (0) 4 minutes ago objective_archimedes
```

Lors de la création du conteneur, nous voyons qu'il a reçu un identifiant unique : **CONTAINER ID**, et un nom en caractère alphanumérique généré aléatoirement : **NAMES**. Nous verrons plus en détail comment configurer le conteneur un peu plus tard.

Attardons-nous sur le **STATUS** qui est **Exited**. Qui signifie que le conteneur a bien démarré une fois, puis s'est éteint.

Cela est normal !

Actuellement, le conteneur n'effectue aucune tâche particulière ; il s'agit simplement d'un environnement dans lequel Node.js est installé. Par défaut, le shell interactif ne nous est pas accessible. Ainsi, plutôt que de rester en fonctionnement sans rien faire, le conteneur s'arrête automatiquement.

Pour modifier ce comportement, nous pouvons créer un nouveau conteneur en utilisant la même commande, mais en ajoutant un nouveau paramètre : **-ti**. Ce paramètre indique que nous souhaitons interagir avec le conteneur.

```
docker container run -ti node
```

```
PS C:\Users\baptiste> docker container run -ti node
Welcome to Node.js v21.7.1.
Type ".help" for more information.
> 1 + 1
2
> |
```

Nous remarquons que la création du nouveau conteneur n'a pas entraîné le téléchargement de l'image ! (Elle est sur notre machine hôte en local maintenant !).

Et nous avons en plus, un prompt dans lequel nous pouvons saisir des commandes **NodeJs** ou **Javascript** qui s'exécuteront seulement à l'intérieur de notre conteneur **et pas dans notre machine hôte, soyons bien claire avec cela !**

Pour sortir du shell, tapez sur la combinaison de touche du clavier : **CTRL + C** deux fois.

Et listons les conteneurs qui ont été créés :

```
docker container ps -a
```

ou

```
docker ps -a
```

Il existe maintenant plusieurs conteneurs Docker, basés sur la même image **Node**, indépendant les uns des autres.

Nous avons pris l'exemple de l'image **Node** pour illustrer le concept de conteneurs et d'images Docker. Cependant, tout ce que nous avons appris reste valable quelque soit votre environnement technique : PHP, PYTHON, RUBY, Etc.



En règle générale, vous utiliserez à chaque fois une image de base officielle pour créer un conteneur. Puis, vous personnaliserez cette image en ajoutant vos propres fichiers, dépendances, etc.

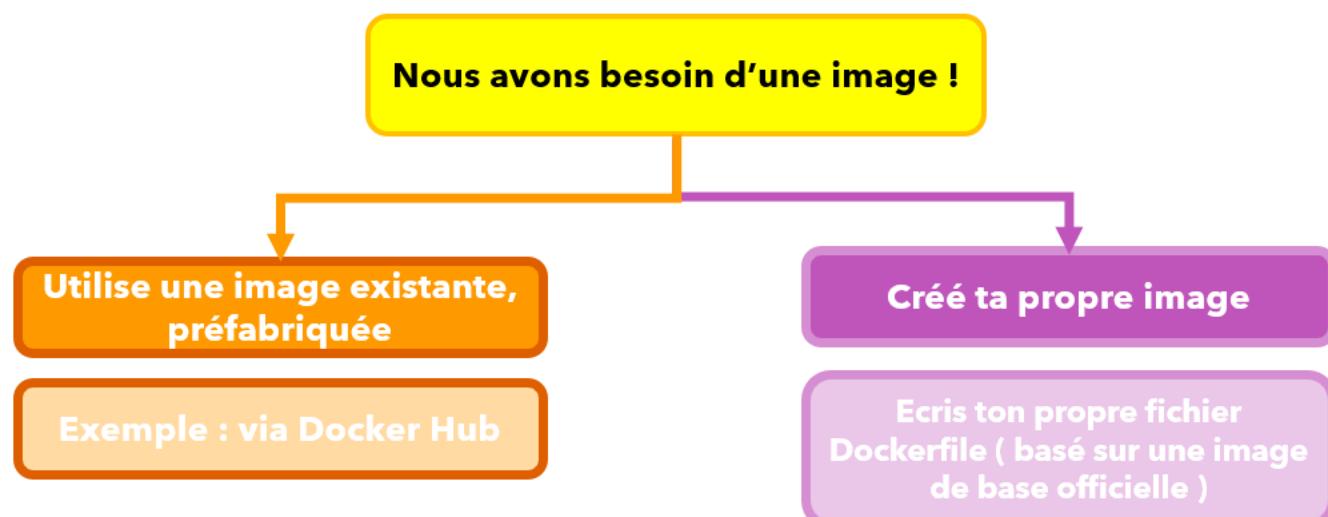
Nous verrons comment créer des images personnalisées dans le chapitre suivant.

3.4. Les images Docker personnalisée

3.4.1. Préambule

Nous avons appris comment utiliser une image **Docker** existante pour créer un conteneur. Cependant, il est souvent nécessaire de créer des images personnalisées pour répondre aux besoins spécifiques de nos applications. Par exemple, si nous souhaitons déployer une application **Node.js**, nous allons utiliser une image de base **Node.js**, puis y ajouter nos fichiers et dépendances.

Dans ce sous-titre, nous allons voir comment procéder en suivant un exemple concret.



3.4.2. TD : Créer une image pour une application Node.js

3.4.2.1. Objectif

Créer une image Docker personnalisée à partir d'une image NodeJs officielle et d'y ajouter le code source d'une application Node.js existante.

3.4.2.2. Préparation

Pour réaliser ce TD, il n'est pas nécessaire de connaître NodeJS, ni Javascript.



Nous allons simplement utiliser Node.js pour illustrer la création d'une image Docker personnalisée.

Nous vous fournirons les fichiers nécessaires pour réaliser ce TD.

- Récupérez le fichier `td01_app_nodejs.zip` dans le répertoire `resources` de ce chapitre.
- Décompressez le fichier dans un répertoire de votre choix.

3.4.2.3. Présentation des fichiers

Le répertoire décompressé contient :

- Un répertoire `public` contenant :
 - Un fichier `styles.css` : une feuille de style simple.
- Un fichier `package.json` : un fichier de configuration pour Node.js.
- Un fichier `server.js` : un fichier Javascript qui crée un serveur web simple.

Le fichier `server.js` contient le code de notre application Node.js. Si vous connaissez Node.js, vous pouvez l'ouvrir pour voir son contenu. Sinon, ne vous inquiétez pas, nous n'aurons pas besoin de le modifier.

Toutefois, voici quelques explications sur ce fichier :

```
1 // Création d'un serveur HTTP avec Express en NodeJS
2 const express = require('express');
3 const bodyParser = require('body-parser');
4 const app = express();
5
6 // Le serveur HTTP démarre et écoute sur le port 80
7 app.listen(80);
```

Le code ci-dessus crée un serveur web simple qui écoute sur le port 80 et nous gérons les requêtes HTTP entrantes (méthodes `GET` et `POST`) pour deux URL différentes : `/` et `/store-goal` :

`server.js` : Code du traitement de la requête HTTP GET

```
// [...Some Code before]
```

```

app.get('/', (req, res) => {
  res.send(`
    <html>
      <head>
        <link rel="stylesheet" href="styles.css">
      </head>
      <body>
        <section>
          <h2>Objectif : </h2>
          <h3>${userGoal}</h3>
        </section>
        <form action="/store-goal" method="POST">
          <div class="form-control">
            <label>Course Goal</label>
            <input type="text" name="goal">
          </div>
          <button>Ajouter un objectif</button>
        </form>
      </body>
    </html>
  );
});

// [...Some Code after]

```

Le code ci-dessus gère la requête HTTP GET pour l'URL `/`. Il renvoie une page HTML contenant un formulaire pour saisir un objectif et affiche aussi l'objectif saisi précédemment ou un objectif par défaut.

server.js : Code du traitement de la requête HTTP POST

```

// [...Some Code before]
app.post('/store-goal', (req, res) => {
  const enteredGoal = req.body.goal;
  console.log(enteredGoal);
  userGoal = enteredGoal;
  res.redirect('/');
});

// [...Some Code after]

```

Le code ci-dessus gère la requête HTTP POST pour l'URL `/store-goal`. Il récupère l'objectif saisi dans le formulaire, le stocke dans une variable `userGoal`, puis redirige l'utilisateur vers la page d'accueil.

Le fichier `package.json` central pour les applications Node.js, car il contient toutes les informations nécessaires pour installer les dépendances de l'application.

Extrait du fichier package.json

```

// [...Some Code before]
{
  "dependencies": {
    "express": "^4.17.1",
  }
}

```

```
"body-parser": "1.19.0"  
}  
// [...Some Code after]
```

Le fichier `package.json` nous montre que cette application nécessite la présence de deux dépendances pour fonctionner : `express` et `body-parser`.



Je ne donnerais pas plus d'explication, ce cours ne traite pas de NodeJs, mais de comment "Dockeriser"/"Conteneuriser" cette application d'exemple.

3.4.2.4. Lancez l'application Node.js en Local

Avant de créer l'image Docker, nous allons lancer l'application Node.js en local pour vérifier qu'elle fonctionne correctement et surtout pour comprendre son fonctionnement.

Pour exécuter une application Node.js, il faut d'abord installer Node.js sur votre machine.

Pour vérifier si Node.js est installé sur votre machine, ouvrez un terminal et tapez la commande suivante :

```
node -v
```

Si Node.js est installé, vous verrez sa version s'afficher. Sinon, vous devrez l'installer.

Rendez-vous sur le lien suivant pour télécharger et installer Node.js sur votre machine :

[Télécharger Node.js](#)

Sélectionnez la version adaptée à votre système d'exploitation.

Une fois Node.js installé, ouvrez **un nouveau terminal** et déplacez-vous dans le répertoire où vous avez décompressé les fichiers de l'application `Node.js`.

Installez les dépendances de l'application en exécutant la commande suivante :

```
npm install
```

```
PS C:\Users\baptiste\Documents\Cours\SPOT_COURS\chapters\docker\images_docker\code\td01_app_nodejs> npm install  
added 81 packages, and audited 82 packages in 2s  
12 packages are looking for funding  
  run 'npm fund' for details  
2 high severity vulnerabilities  
To address all issues, run:  
  npm audit fix --force  
Run 'npm audit' for details.
```

Répertoire contenant les fichiers et dossiers de l'application NodeJS

Un nouveau dossier `node_modules` est créé dans le répertoire de l'application. Il contient toutes les dépendances nécessaires pour exécuter l'application.

Nom	Modifié le	Type	Taille
public	02/04/2024 00:17	Dossier de fichiers	
package.json	02/04/2024 00:15	Fichier source JSON	1 Ko
server.js	02/04/2024 01:24	Fichier source Jav...	2 Ko
node_modules			
package-lock.json	02/04/2024 01:50	Fichier source JSON	32 Ko

Nouveau répertoire contenant les dépendances

Vous pouvez maintenant lancer l'application en exécutant la commande suivante :

```
node server.js
```

Laissez le terminal ouvert et ouvrez un navigateur web.

Tapez l'URL <http://localhost> dans la barre d'adresse pour accéder à l'application.

Objectif :

Apprendre Docker!

Objectif de ce cours :

Ajouter un objectif

Testez l'application en saisissant un objectif dans le champ de texte et en cliquant sur le bouton **Ajouter un objectif** et observez le résultat.

Cette application fonctionne donc localement sans Docker !

Maintenant, arrêtons le server Node.js en tapant **CTRL + C** dans le terminal. Et supprimons le dossier **node_modules** en tapant la commande suivante :

Sous Windows

```
rm node_modules
```

Sous Linux

```
sudo rm -R node_modules
```

et supprimons le fichier `package-lock.json` :

```
rm package-lock.json
```

Nous allons maintenant créer une image Docker spécialement pour cette application !

3.4.2.5. Création de notre propre image Docker

3.5. Exercices

Unresolved directive in/src/includes/add_exercise.adoc
include:../../chapters/docker/images_docker/exercises/app_nodejs/main.adoc[]