

Docker : Maîtrisez la conteneurisation applicative de A à Z

Table des matières

Découvrons d'abord la notion de conteneur	3
Docker : qu'est-ce que c'est ?	3
Quelle est son histoire ?	4
Comment fonctionne-t-il ?	4
Pourquoi utiliser Docker ?	6
Quels sont les composants de Docker ?	7
Docker Engine	7
Docker Daemon	7
Docker Client	7
Docker CLI	7
Docker Machine	8
Docker Desktop	9
Docker Registry	10
Docker Hub	10
Dockerfile	10
Docker Image	11
Docker Container	11
Docker Volume	11
Docker Compose	11
Docker Swarm	11
Kubernetes	12
Réseau Docker	12
Docker Nginx	12
Tutoriel : Prise en main et utilisation de Docker	13
Télécharger Docker	13
Installer Docker	14
Sur Windows	14
Pour MacOS	18
Pour Linux/Ubuntu	19
Lancer votre premier conteneur	19

Télécharger une image depuis le Docker Hub.....	22
Arrêter un conteneur	23
Supprimer un conteneur	23
Supprimer une image	23
Créer un Dockerfile.....	24
Créer un fichier .dockerignore	25
Construire votre propre image.....	26
Exécuter votre propre conteneur docker.....	26
Lancer une application dans un serveur Nginx	27
Partager une image	30
Partager sur Docker Hub	30
Exporter une image	31

Nous avons vu dans notre article sur les [8 métiers porteurs du Big Data](#) que les ingénieurs DevOps/Cloud et les développeurs Big Data font partie des postes les plus recherchés en ce moment dans ce domaine. De ce fait, si l'on souhaite s'orienter vers ces types de métiers, il est important de connaître et de maîtriser toutes les technologies nécessaires pour les pratiquer. Docker fait partie de ces derniers, car il permet d'optimiser leurs travaux de manière considérable.

Dans cet article, nous allons voir ce qu'est Docker, ensuite nous allons donner un tutoriel exhaustif qui vous permettra de mieux le prendre en main.

Découvrons d'abord la notion de conteneur

Avant d'entrer dans le vif du sujet, il est important de comprendre la notion de conteneur ou container en anglais.

Un conteneur est donc un environnement permettant d'isoler des applications et leurs dépendances respectives afin d'éviter que celles-ci ne créent des interférences entre elles. Il s'agit d'une alternative aux machines virtuelles utilisées auparavant qui, elles, vont simuler un système entier. Les conteneurs, quant à eux, vont seulement puiser les ressources nécessaires à l'exécution de l'application dans le Kernel de l'OS principal de la machine physique. Cela permet donc de simuler plusieurs systèmes sur le même OS.

Un conteneur peut être relié à d'autres conteneurs grâce à des systèmes de réseaux virtuels. Les ressources dédiées à chaque conteneur peuvent être augmentées selon le besoin de l'application. Et le conteneur peut être stoppé dès qu'aucun processus n'est en cours d'exécution. Tout cela fait que les conteneurs sont plus performants, plus flexibles, plus rapides, plus légers et plus simples à mettre en place et à utiliser.

Docker : qu'est-ce que c'est ?

Maintenant que l'on sait ce qu'est un conteneur, découvrons Docker. Alors, Docker est la plateforme de conteneurisation la plus populaire et la plus utilisée de nos jours. Il reprend toute la notion de conteneur, c'est-à-dire qu'il sépare des applications et leurs dépendances dans des conteneurs. On peut créer, tester, déployer et exécuter des applications sur n'importe quel environnement à l'aide de Docker. Cela évite les éventuels conflits au niveau de chaque composant et donc de résoudre les soucis de version, d'incompatibilité ou encore d'exécution qui surviennent souvent lorsque l'on développe une solution.

Docker est une solution open source développée par Docker Inc. et de nombreux contributeurs rassemblés dans la communauté Docker. Il est basé sur le Kernel Linux et étend le format de conteneur LXC (Linux Container) pour ensuite basculer sur libcontainer, sa propre bibliothèque depuis 2015. Il permet également de lancer des conteneurs sur les systèmes d'exploitation Windows et Mac. Docker peut être déployé sur le Cloud, raison pour laquelle de géants de ce domaine tels que Google, Microsoft ou Amazon ont décidé de l'implémenter au sein de leurs systèmes.

Il offre un réel avantage aux systèmes distribués, car ces derniers peuvent être exécutés individuellement sur un ordinateur physique ou sur une instance de nœud. Cela offre une énorme flexibilité en matière de ressources allouées à chaque nœud de cluster.

Docker est un outil intéressant pour les ingénieurs DevOps/Cloud et les développeurs Big Data de par sa fiabilité, sa sécurité, sa simplicité, sa moindre consommation de ressources, sa rapidité et sa scalabilité, qui sont des critères importants lorsque l'on travaille sur une énorme quantité de données.

En plus de tout cela, il permet également de travailler avec de nombreux langages, framework et technologies tels que [Node.js](#), ASP.Net, Ruby, PHP, [Java](#), WordPress, PostgreSQL, etc.

Quelle est son histoire ?

En 2010, Solomon Hykes qui travaillait à ce moment-là chez dotCloud, une entreprise française proposant des solutions de PaaS (Platform as a Service), et deux de ces collègues, Andréa Luzzardi et François-Xavier Bourlet, ont décidé de fonder Docker, l'entreprise, et ont lancé le projet Docker, la plateforme.

Le projet est devenu open source à partir de 2013, ce qui a contribué à la popularisation du système de conteneurisation pour devenir de nos jours la plateforme la plus utilisée dans ce domaine.

En 2019, l'entreprise Mirantis rachète la version Enterprise de Docker et reprend donc les clients et les employés de ce dernier.

Nous sommes actuellement à la version 4.5.x de Docker Desktop pour Windows et MacOS, à la version 20.10 de Docker Engine et 1.29.x en ce qui concerne Docker Compose. Ne vous inquiétez pas, nous allons découvrir ces outils un peu plus tard dans l'article.

Comment fonctionne-t-il ?

Avant, on utilisait exclusivement des machines virtuelles pour faire fonctionner des applications sous un autre système d'exploitation. Les machines virtuelles utilisent un hyperviseur afin de distribuer les ressources matérielles de la machine physique pour mettre en place un système d'exploitation complet.

Cependant, ce procédé coûte cher en matière de ressources, car elles doivent être prédéfinies à l'avance, ce qui pose quelques problèmes au niveau de l'extension des applications. Et c'est là qu'interviennent les plateformes de conteneurs comme Docker.

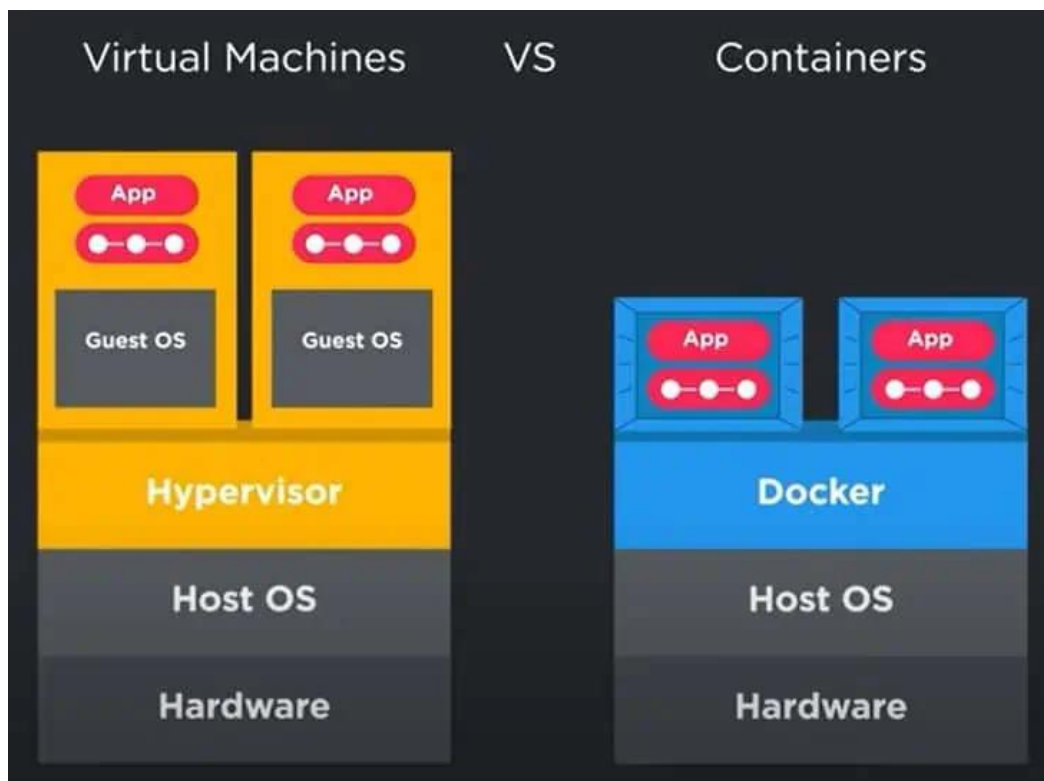
Docker s'appuie sur LXC, ce qui veut dire que certains composants de ce système de conteneur Linux sont utilisés dans Docker à savoir les **cgroup**, les **namespaces** ou encore **selenux**. Ajouté à cela, il utilise des bibliothèques qui lui sont propres comme **libcontainer**, **libchan** ou encore **libswarm**.

cgroup est le groupe de contrôle permettant de distribuer les ressources nécessaires à chaque processus. Les **namespaces** sont des espaces de nom à appliquer sur les processus offrant ainsi une barrière entre chacun d'eux. Et **selenux** est un système de sécurité qui vérifie les droits utilisateurs.

Libcontainer est la bibliothèque utilisée par Docker pour gérer les conteneurs, c'est-à-dire les créer et effectuer des opérations sur ces derniers. **Libchan** est celle qui sert à établir la communication entre les réseaux virtuels de Docker et **Libswarm** est celle qui contient les outils nécessaires à la création de ces réseaux.

Ces dernières, en plus d'autres bibliothèques, permettent d'isoler chaque processus afin qu'ils puissent s'exécuter de manière autonome. Et ce, en limitant les ressources allouées à chacun d'eux et en optimisant leurs performances, contrairement à une VM traditionnelle.

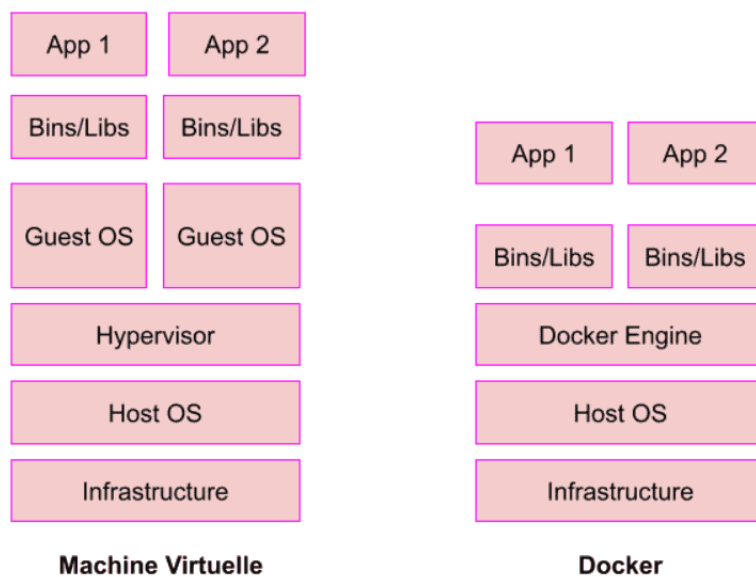
Les conteneurs Docker sont déployés sous forme d'**images**. C'est à partir de ces images de conteneurs que l'on crée le conteneur sur une plateforme hôte, qu'on l'exécute sur celle-ci et qu'on le partage sur Docker Hub, par exemple.



Docker vs machine virtuelle

Pourquoi utiliser Docker ?

Tout d'abord, on va effectuer un bref comparatif entre une machine virtuelle et Docker.



Comme vous pouvez le voir sur cette image, chaque VM possède son propre système d'exploitation. De ce fait, elle est plus difficile à mettre en place et consomme beaucoup de ressources. Ce n'est pas le cas de Docker, car le **Docker Engine** se pose juste au-dessus du système d'exploitation de l'hôte. Donc, les premiers avantages de Docker sont qu'il est plus facile à mettre en place et consomme moins de ressources que son prédécesseur.

En plus de cela, cette structure permet de lancer plus de conteneurs, donc plus d'applications, sur une même infrastructure. L'évolutivité que propose Docker est donc plus importante par rapport à l'utilisation des machines virtuelles.

Docker possède d'autres points forts, mais en résumé, voici les raisons pour lesquelles on l'utilise :

- On peut exécuter plusieurs applications en utilisant différents OS sur une même machine hôte, sans avoir à créer les systèmes en entier ;
- Les conteneurs peuvent être migrés n'importe où en générant leurs images une seule fois ;
- Docker est multiplateforme, on peut l'exécuter sur Windows, MacOS et Linux, bien sûr ;
- Les langages les plus populaires et les plus utilisés sont pris en charge par lui ;
- Il est léger, rapide, performant, facile à utiliser et moins gourmand en ressources ;
- Les applications conteneurisées sont faciles à déployer, à exécuter, à maintenir et à partager ;
- La gestion des ressources utilisées par les applications est moins fastidieuse ;
- Grâce à la superposition des couches des images de conteneurs Docker, on peut facilement gérer les versions d'une application et par conséquent revenir à une version antérieure lors d'un éventuel dysfonctionnement.

Quels sont les composants de Docker ?

Nous entrons petit à petit dans la partie technique de Docker. Avant de commencer le tutoriel, nous allons voir les composants de bases qu'il faut connaître si l'on souhaite utiliser Docker.

Docker Engine

Docker Engine est le système qui fait tourner l'ensemble de Docker. Il s'agit d'un moteur qui suit une architecture client-serveur comprenant principalement trois composants : **Docker Daemon**, **Docker CLI** (nous allons développer ces deux composants juste en bas) et une API qui sert entre autres d'intermédiaire entre les deux composants précédents.

Ses principaux rôles sont de gérer chaque processus de création, d'exécution des applications dans les conteneurs. Il crée les processus Docker Daemon afin que tous les composants de Docker puissent fonctionner parfaitement.

Pour garantir l'état de Docker à un moment donné, il faut le paramétrer, c'est-à-dire déclarer les conditions requises et les paramètres nécessaires à votre plateforme.

Pour l'installation et la gestion de Docker Engine sur des hôtes virtuels ou sur les anciennes versions des OS supportés par Docker, il faut utiliser un autre composant à savoir **Docker Machine**.

Plusieurs plug-ins peuvent être installés sur Docker Engine, on retrouve ces derniers sur les **Docker Registry**.

Docker Daemon

Docker Daemon est la partie serveur de Docker Engine. Il agit comme un système d'exploitation, c'est-à-dire qu'il gère les processus et les objets Docker comme les images, les volumes, les conteneurs et les réseaux. Pour ce faire, il écoute les requêtes envoyées par l'API Docker et exécute ce qui est demandé par cette dernière.

Docker Daemon peut également communiquer avec d'autres Docker Daemon afin de gérer les services.

Docker Client

Comme son nom l'indique, il s'agit de la partie cliente de l'architecture globale de Docker, c'est-à-dire la partie où les utilisateurs interagissent avec le démon Docker. C'est lui qui envoie les requêtes de l'API en interprétant les commandes tapées par l'utilisateur afin que Docker Daemon puisse les exécuter. Un Docker Client peut communiquer avec plusieurs Docker Daemon.

Docker CLI

Docker CLI est l'abréviation de Docker Command Line Interface. Il s'agit des commandes à exécuter lors de chaque opération que l'on effectue sur Docker. Il existe trois types de CLI :

- Les **Docker CLI** sont les CLI principaux de Docker. Pour obtenir la liste de toutes les commandes disponibles, il suffit de taper **docker** ou **docker help** sur une invite de commande ;
- Les **Compose CLI** qui sont les CLI pour interagir avec **Docker Compose** ;

- Les **Daemon CLI** sont les CLI permettant de gérer les conteneurs et leurs processus.

Voici quelques exemples de commandes docker :

- Connaître la version de Docker installé :

```
docker -- version
```

- Lancer un conteneur :

```
docker run -d -p 80:80 docker/getting-started
```

- Lister tous les conteneurs :

```
docker ps -a
```

- Obtenir toutes les images présentes sur le système :

```
docker images
```

- Construire une image à partir d'un Dockerfile :

```
docker build
```

- Supprimer un conteneur :

```
docker rm ID_du_conteneur
```

- Supprimer une image :

```
docker rmi ID_de_l_image
```

- Afficher les logs d'un conteneur avec leurs détails :

```
docker logs --details
```

Nous allons en découvrir davantage dans la partie tutoriel de l'article. Mais si vous souhaitez connaître toutes les commandes et la manière de les utiliser, vous pouvez consulter la [liste des commandes](#) dans la documentation officielle de Docker.

Docker Machine

Avant toute chose, il faut savoir que Docker Machine est maintenant considéré comme obsolète par Docker Inc. et son code source a été archivé en conséquence. Toutefois, il est intéressant de le connaître, car il peut servir dans certains cas, comme nous l'avons mentionné auparavant.

L'évènement de Docker Desktop pour les systèmes d'exploitation Windows et MacOS qui embarque déjà la dernière version de Docker Machine a entraîné son déclassement. En effet, dans ces deux versions de Docker, on peut créer des machines virtuelles grâce à **VirtualBox** pour les Mac et **HyperV** pour les systèmes Windows. Et si le système d'exploitation sur l'un de ces deux appareils est d'une

version moins récente, on peut utiliser **Docker ToolBox** pour cela. Mais si vous souhaitez obtenir Docker Engine, sachez qu'il est disponible sur GitHub.

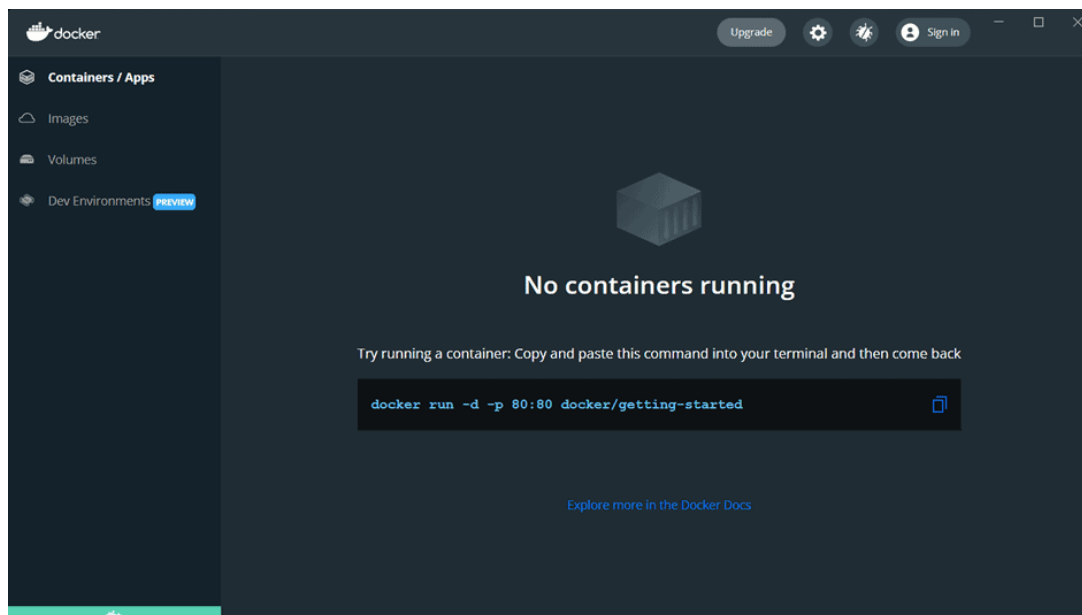
Vous l'aurez sans doute compris, Docker Machine permet de créer une machine virtuelle sur un ordinateur physique afin que l'on puisse utiliser Docker à l'intérieur de celle-ci. Il offre la possibilité de gérer Docker Engine à distance, de le mettre à jour, de visualiser son état et même de redémarrer la machine virtuelle, selon le pilote de cette dernière.

L'utilisation de Docker Machine peut être importante, surtout lorsque l'on n'a pas la possibilité de se procurer de moyen de se procurer les environnements adéquats. C'est une alternative peu coûteuse pour effectuer des tests avant le déploiement final d'une application. Il permet aussi de mieux appréhender le passage vers le cloud et de créer l'ensemble de ressources nécessaires au conteneur.

Docker Desktop

Docker Desktop est le logiciel Docker que l'on peut installer sur Windows et MacOS. Il nous permet de pouvoir utiliser Docker en passant par une interface intuitive, d'autant plus que son installation est très facile. Lorsque l'on installe Docker Desktop, plusieurs composants sont directement inclus tels que Docker Engine, Docker Compose, Docker CLI, etc. Et on peut télécharger d'autres composants en se rendant sur les Docker Registries.

Vous avez la possibilité de choisir entre 3 versions de Docker Desktop : le **Docker Desktop**, le **Docker Community Edition** et le **Docker Enterprise Edition** dont les deux premières sont gratuites. Quant à la dernière version, elle est destinée aux entreprises employant plus de 250 personnes ou générant plus de 10 millions de dollars de revenus annuels. Ces entreprises doivent donc souscrire à un abonnement payant pour pouvoir utiliser Docker Desktop.



Dans la partie tutoriel de cet article, nous allons utiliser Docker Desktop, la première version, afin d'illustrer toutes les pratiques que nous allons effectuer.

Tutoriel Docker – Partie 1

Réalisation : Guillaume DELACROIX Formateur AFPA
30 août 2023

Afpa

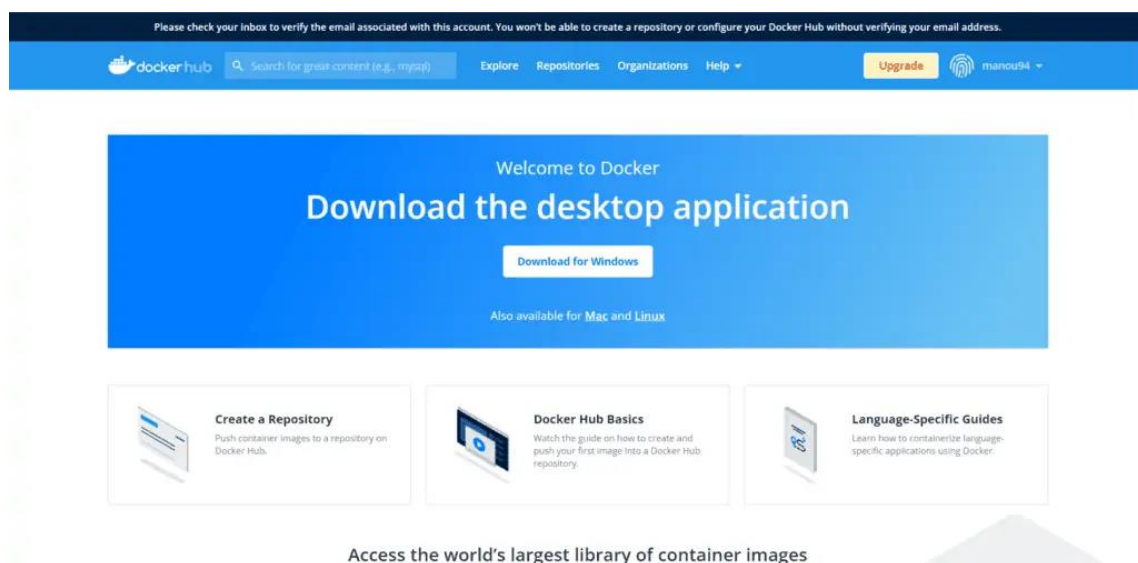
Docker Registry

Docker Registry est l'emplacement où l'on peut stocker et télécharger des images Docker. Les images stockées sur ces registres disposent d'un système de versionning permettant aux utilisateurs de suivre l'évolution des images en question.

Il existe deux types de Docker Registry : les registres publics et les registres privés. Le registre par défaut de Docker est Docker Hub, qui permet d'accéder à des images, mais également de créer des registres privés. D'autres registres sont également accessibles, comme GitHub, par exemple. Vous pouvez même créer votre propre Docker Registry si vous le souhaitez.

Docker Hub

Comme nous l'avons vu précédemment, Docker Hub est le registre par défaut de Docker. C'est un hébergeur d'images de conteneurs qui contient plus 100 000 images dont certaines ont été développées et est maintenu par Docker Inc. lui-même. On y retrouve également des images partagées par des développeurs tiers et on peut les télécharger pour nos propres besoins de conteneurisation.



Tous ses utilisateurs peuvent effectuer ce téléchargement et ce partage d'image et il est aussi intégré à GitHub. Si l'on connecte un compte Docker Hub à Docker Desktop, on bénéficie de certains avantages tels que l'augmentation des tirages qui devient deux fois plus nombreuse par rapport à un utilisateur anonyme.

Dockerfile

Dockerfile est la base sur laquelle tous les conteneurs sont construits. Il contient l'ensemble des commandes à exécuter afin de construire l'image à savoir le système d'exploitation de base du conteur, les langages utilisés, l'environnement et ses variables, l'emplacement des fichiers, etc. La création de Dockerfile se fait à l'aide des commandes CLI.

Le but de ces déclarations est de pouvoir automatiser la création de l'image du conteneur par la suite.

Docker Image

Une fois le Dockerfile écrit, on peut maintenant créer l'image d'un conteneur. Une image est un modèle à lecture seule pour les conteneurs. On y retrouve tout le code source de l'application à conteneuriser, ses dépendances, les bibliothèques à utiliser ainsi que d'autres outils nécessaires à la création du conteneur.

Elle est composée de plusieurs couches faisant référence aux versions de l'image. C'est-à-dire que lorsque l'on modifie l'image, une nouvelle couche se pose sur les anciennes versions. Donc au final, un Docker image comprend une couche de base à lecture seule et des couches supérieures sur lesquelles on peut écrire.

On peut créer notre propre image, mais dans la plupart des cas, on en télécharge sur le Docker Hub qui va servir de base pour les personnalisations que l'on souhaite effectuer.

Docker Container

Le conteneur, que nous ne cessons d'évoquer depuis le début de cet article, est en réalité une instance d'image sur laquelle on peut interagir. Lorsque l'on exécute une image, cela génère un conteneur. On peut également créer plusieurs conteneurs avec une seule image. Ces derniers peuvent être à leur tour lancés, modifiés, stoppés ou supprimés. Les paramètres du conteneur ainsi que ses conditions d'exécution sont modifiables selon le besoin de l'utilisateur à l'aide des commandes CLI ou d'API.

Docker Volume

Le Docker Volume est l'emplacement où l'on stocke les données utilisées par les conteneurs et celles générées par Docker. Ces données persisteront sur les conteneurs qui y font appel. Cela veut dire que les volumes n'interfèrent pas dans le cycle de vie du conteneur. En plus, on peut les utiliser même si l'on exécute un conteneur Windows ou Linux.

Docker Compose

Docker Compose est l'outil Docker qui sert à créer et gérer l'architecture d'une application multiconteneurisée, c'est-à-dire une application dont les composants tels que le code, la base de données, etc. sont éparpillés sur plusieurs conteneurs.

Pour effectuer cette tâche, Docker Compose utilise YAML pour spécifier les caractéristiques de l'architecture pour que l'on puisse par la suite exécuter tous les conteneurs avec une seule commande.

Docker Swarm

Docker Swarm est un outil d'orchestration pour les conteneurs Docker. Grâce à lui, on peut gérer des conteneurs présents sur les machines hôtes qui composent une architecture distribuée. Ce qui offre une haute disponibilité des applications conteneurisées.

Il sert entre autres à gérer les ressources utilisées par chaque nœud de cluster pour que ces derniers fonctionnent correctement.

Docker Swarm est préinstallé avec le logiciel Docker, c'est-à-dire que l'on peut directement l'utiliser une fois que ce dernier est installé sur tous les hôtes. Il agit également comme un équilibreur de charge sur l'ensemble de l'architecture.

Kubernetes

Kubernetes n'est pas un composant Docker à proprement parler, mais se présente comme une alternative à Docker Compose et Docker Swarm. Il s'agit d'un logiciel open source permettant d'orchestrer des conteneurs. Il permet d'automatiser des tâches relatives aux conteneurs tels que leur déploiement, leur mise à jour, la surveillance d'état ou encore le rééquilibrage des charges.

L'utilisation de Kubernetes est intéressante et devient presque nécessaire surtout lorsque le nombre de conteneurs augmente de manière conséquente. Il permet également de faciliter l'utilisation de Docker dans les environnements distribués, ce qui est le cas lorsque l'on traite du [Big Data](#). En plus, grâce à l'ensemble des outils embarqués dans Kubernetes, on peut mettre en place des Platform-as-a-Service et des Container-as-a-Service performants pour les applications conteneurisées.

Réseau Docker

Il faut savoir qu'on a la possibilité de connecter les conteneurs entre eux, c'est d'ailleurs l'une des raisons qui fait la puissance de Docker. Quel que soit le système d'exploitation utilisé, il existe des moyens afin de gérer les conteneurs reliés, et ce, quel que soit l'état de ces derniers.

Il existe plusieurs types de pilotes réseau Docker à savoir :

- **bridge** qui est le pilote par défaut de Docker ;
- **host** qui supprime l'isolation entre le conteneur et le système de l'hôte ;
- **overlay** qui permet de mettre en place des réseaux superposés afin que l'on puisse connecter plusieurs Docker Engine ou une architecture utilisant Docker Swarm ;
- **ipvlan** qui donne l'accès aux utilisateurs sur l'adressage IPV4 ou IPV6 ;
- **macvlan** qui permet de donner une adresse MAC à un conteneur ;
- **none** qui désactive tous les réseaux.

Docker Nginx

Nginx, que l'on prononce "engine-x", est un logiciel open source sous licence 2-clause BSD qui peut prendre plusieurs rôles. En effet, il peut être un serveur proxy inverse, un serveur web, un cache HTTP, un proxy de messagerie et un équilibreur de charge sur lequel plusieurs serveurs web s'appuient.

C'est en sa qualité de serveur web que Docker a décidé de l'intégrer dans sa plateforme, car il permet de développer des applications web et de les héberger, quel que soit le système d'exploitation utilisé.

On peut retrouver l'image Nginx sur Docker Hub si l'on souhaite l'utiliser. Il vous suffit d'effectuer un pull de cette image, de lancer le conteneur Nginx et de lancer votre application à partir de ce conteneur.

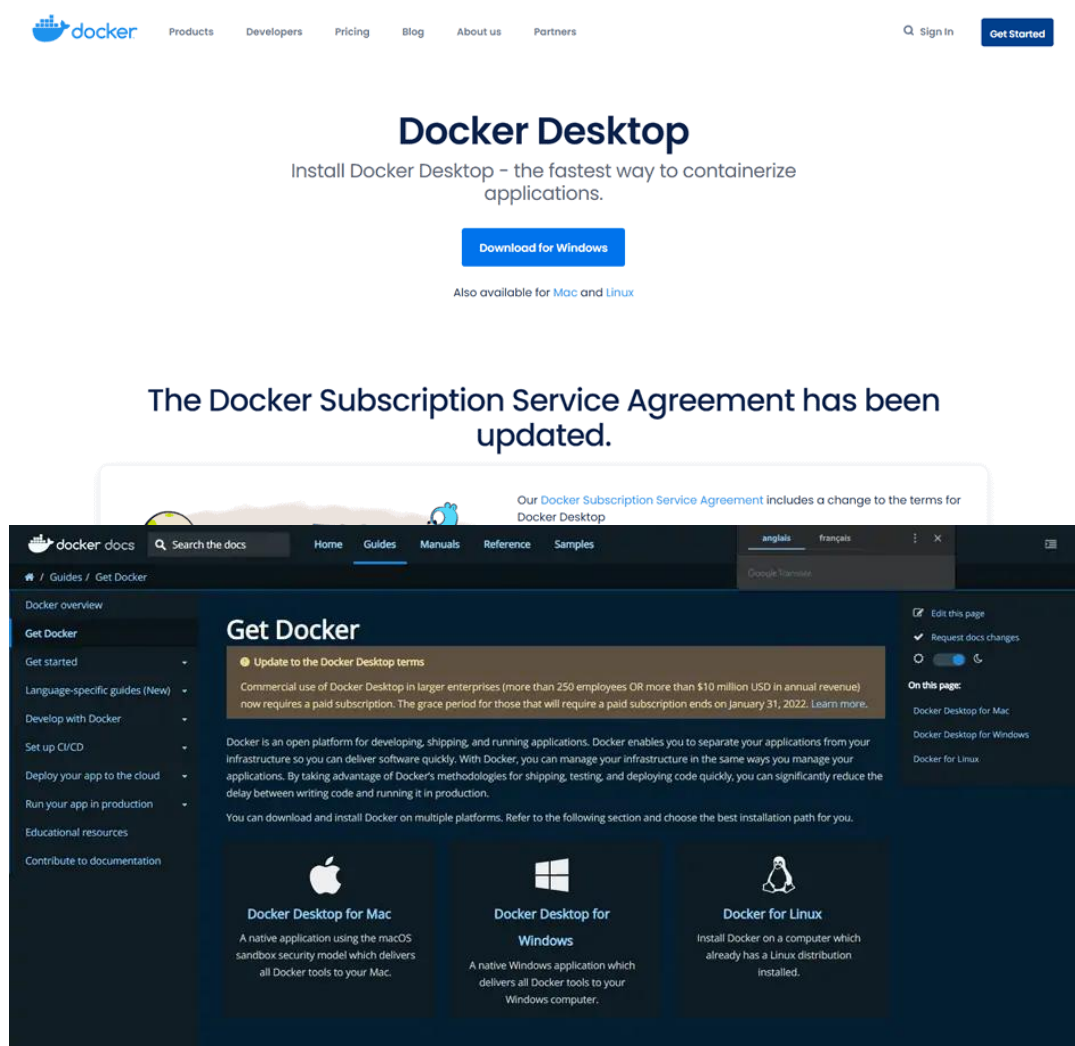
Maintenant que nous avons toutes les informations nécessaires sur Docker, passons à l'application à travers le tutoriel qui suit.

Tutoriel : Prise en main et utilisation de Docker

Dans cette section, nous allons voir étape par étape la manière d'utiliser Docker. Une fois ce tutoriel achevé, vous serez capable d'installer Docker, de manipuler des conteneurs, un dockerfile et des images. Vous allez également voir comment manipuler Docker Compose.

Télécharger Docker

Avant de pouvoir installer Docker, il faut évidemment le télécharger. Vous avez le choix entre deux sites pour effectuer cela : le [site officiel](#) ou [la documentation officielle](#).



Une fois sur l'une de ses pages, on choisit la version qui convient à notre système d'exploitation.

Note :

- Pour pouvoir utiliser Docker Desktop sur Windows, il faut disposer de Windows Professional ou Windows Enterprise, sinon on doit utiliser Docker Toolbox.

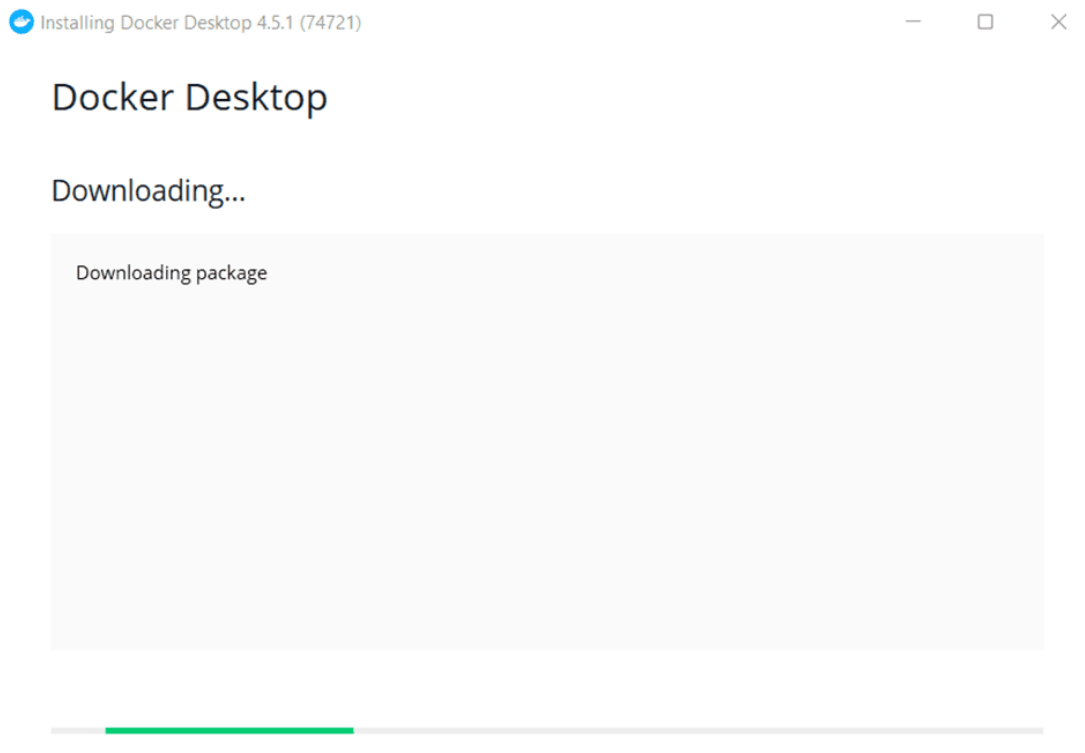
- Si vous avez un Mac, vous devez choisir entre deux versions. En effet, le fichier que vous devez prendre doit correspondre au processeur de votre ordinateur à savoir une puce Intel ou Apple.



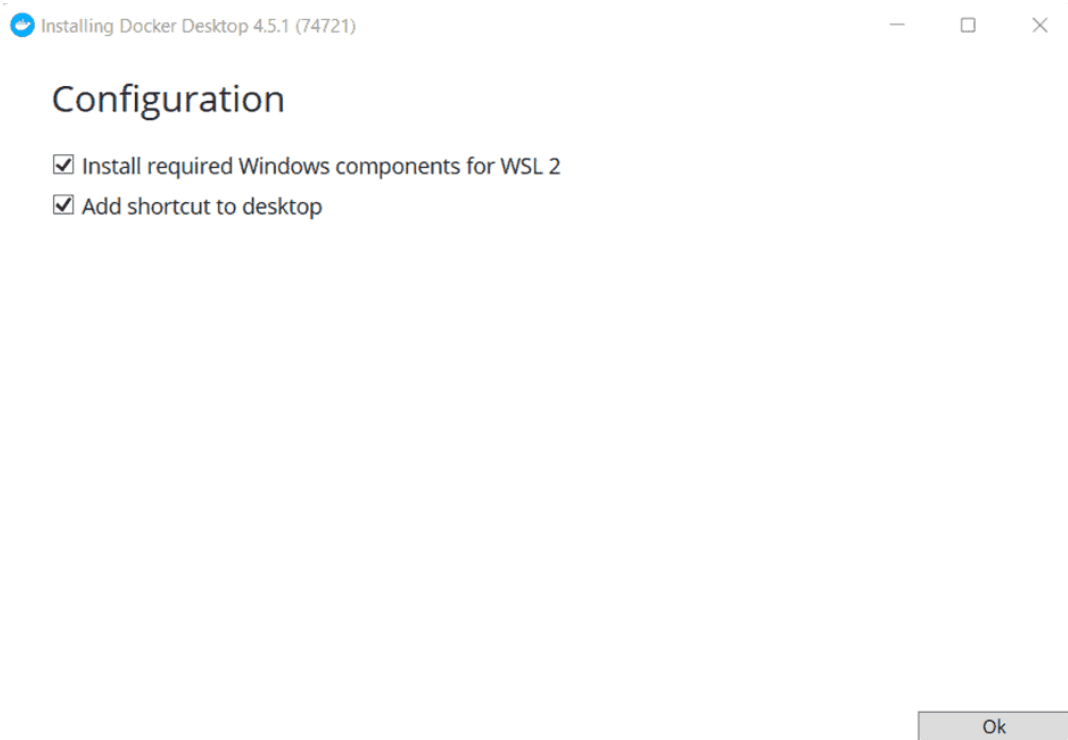
Installer Docker

Sur Windows

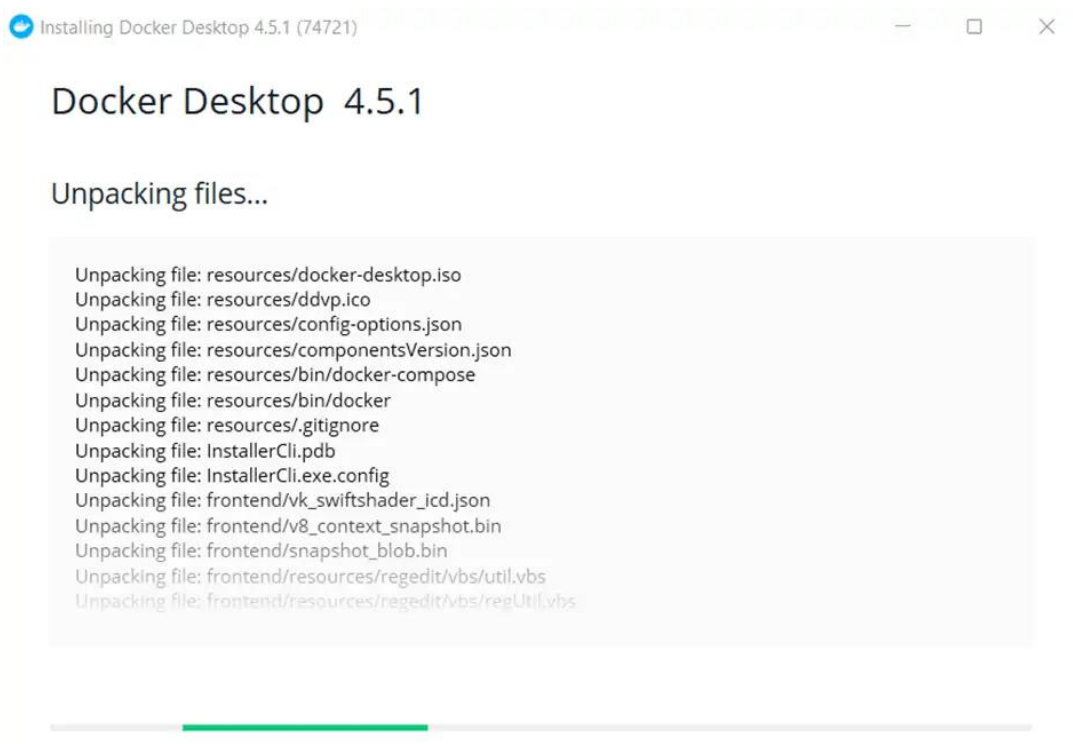
Une fois que vous avez téléchargé la version de Docker qui vous convient, il faut lancer l'installation et pour cela, lancez le fichier exécutable obtenu. Voici ce que vous allez voir une fois que vous avez effectué cela :



Après le téléchargement du package montré sur cette image, vous allez voir la fenêtre d'installation qui suit :



Cliquez sur Ok pour lancer l'installation à proprement parler :



Dès que cette étape s'achève, vous avez terminé l'installation :

Docker Desktop 4.5.1

Installation succeeded

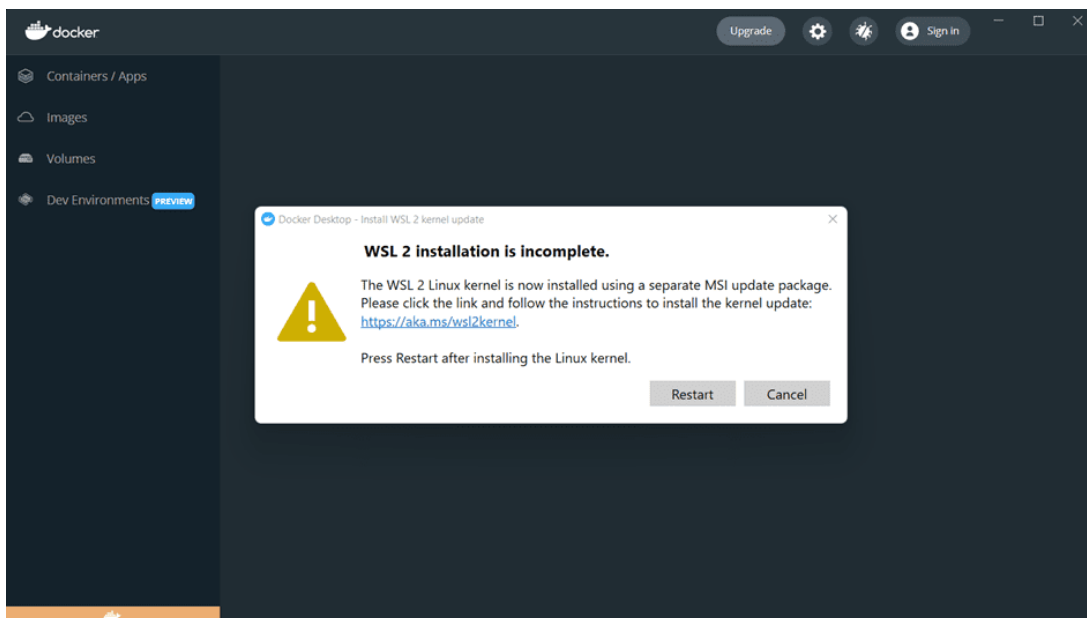
You must restart Windows to complete installation.

Close and restart

Vous constatez qu'il demande un redémarrage de votre PC, effectuez-le et vous pourrez utiliser Docker sur votre machine.

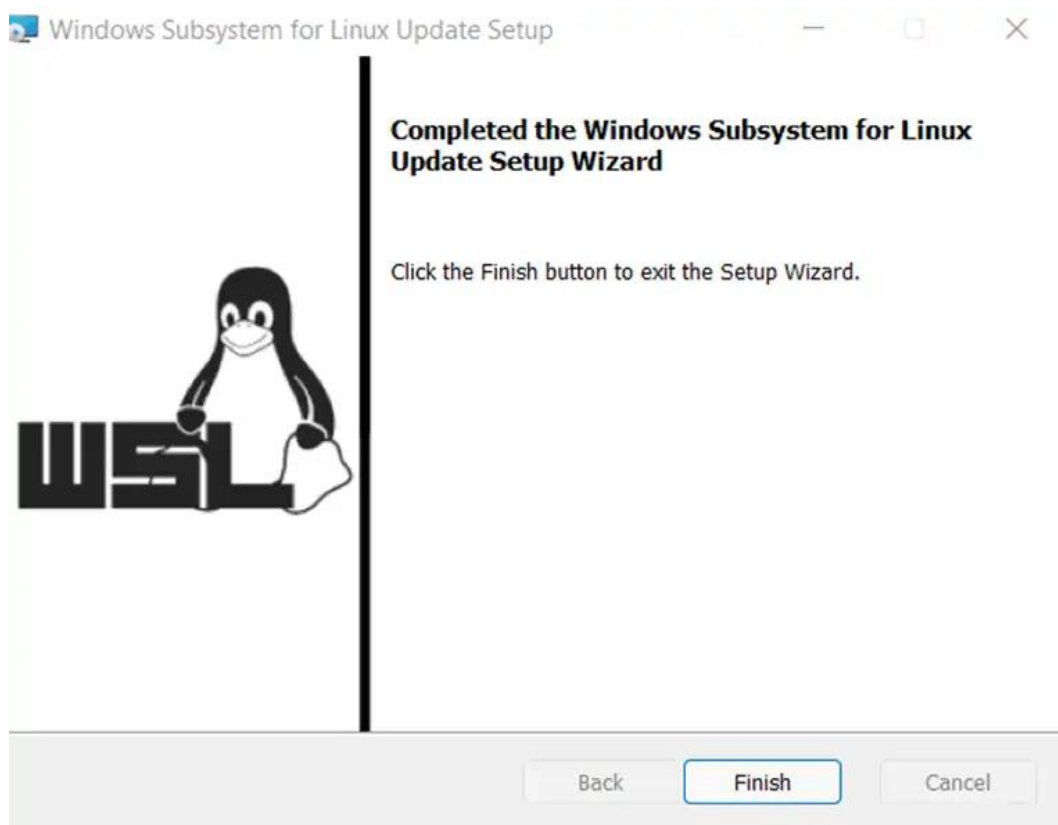
Note :

Si vous ne disposez pas encore de noyau Linux sur votre machine, il se peut que vous ayez ce message qui vous demande de le mettre à jour :

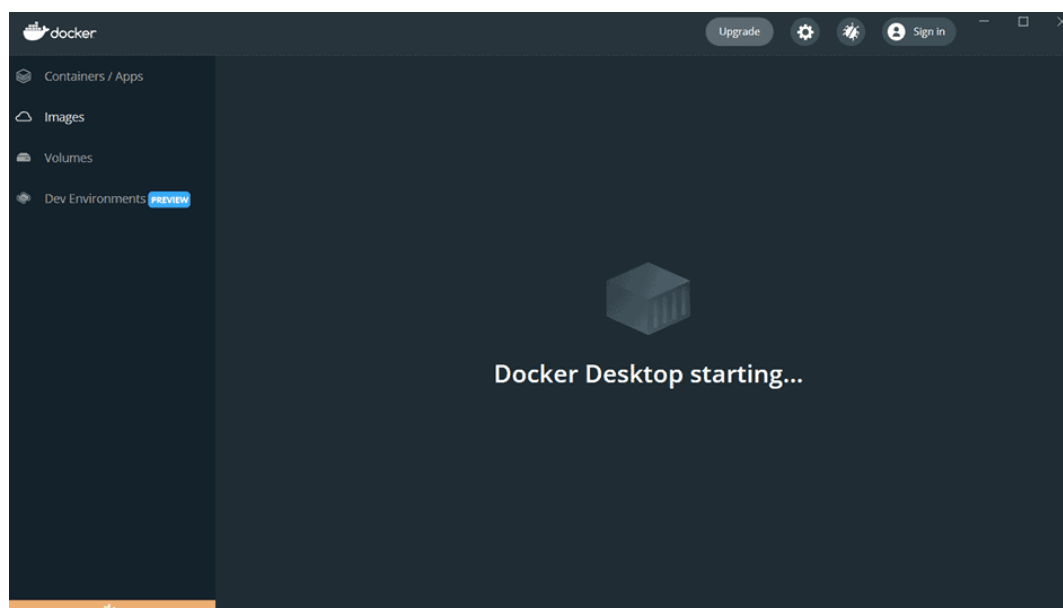


Comme le message le mentionne, Docker a installé le kernel Linux, mais dans un MSI séparé.

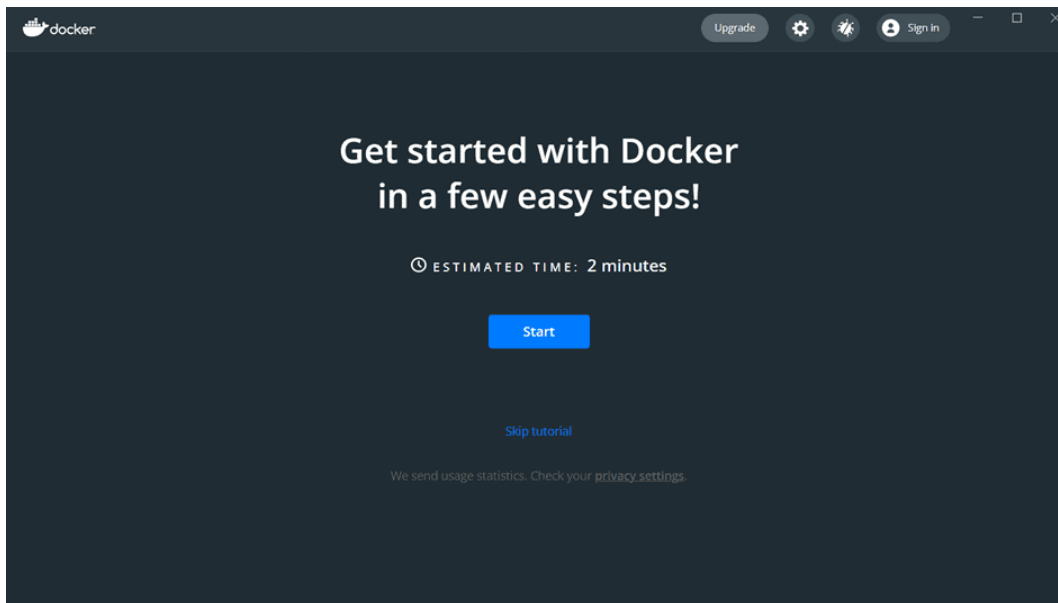
Pour que tout soit en ordre, téléchargez le noyau demandé en suivant le lien qu'il propose et lancez l'installation comme suit :



Relancez Docker après cette installation :



Et voilà, tout est prêt pour que vous puissiez commencer.



Pour MacOS

L'installation pour MacOS est assez simple. En effet, il suffit de lancer l'exécutable, d'attendre les vérifications qu'effectue votre ordinateur puis de glisser Docker dans votre dossier d'application.



Pour Linux/Ubuntu

Avant de lancer l'installation, il faut installer tous les packages Ubuntu nécessaires pour utiliser Docker correctement. Pour ce faire, lancez les commandes suivantes :

```
sudo apt-get install apt-transport-https ca-certificates curl software-properties-common
```

Ensuite, ajoutez la clé GPG du dépôt Docker sur la machine avec la commande ci-dessous :

```
curl -fsSL https://download.docker.com/linux/ubuntu/gpg | sudo apt-key add -
```

Dès que c'est fait, ajoutez maintenant le référentiel Docker APT :

```
sudo add-apt-repository "deb [arch=amd64] https://download.docker.com/linux/ubuntu focal stable"
```

Après, il faut mettre à jour les sources APT contenant les packages Docker avec la commande suivante :

```
sudo apt update
```

Maintenant, on peut passer à l'installation de Docker à proprement parler ainsi que des paquets supplémentaires :

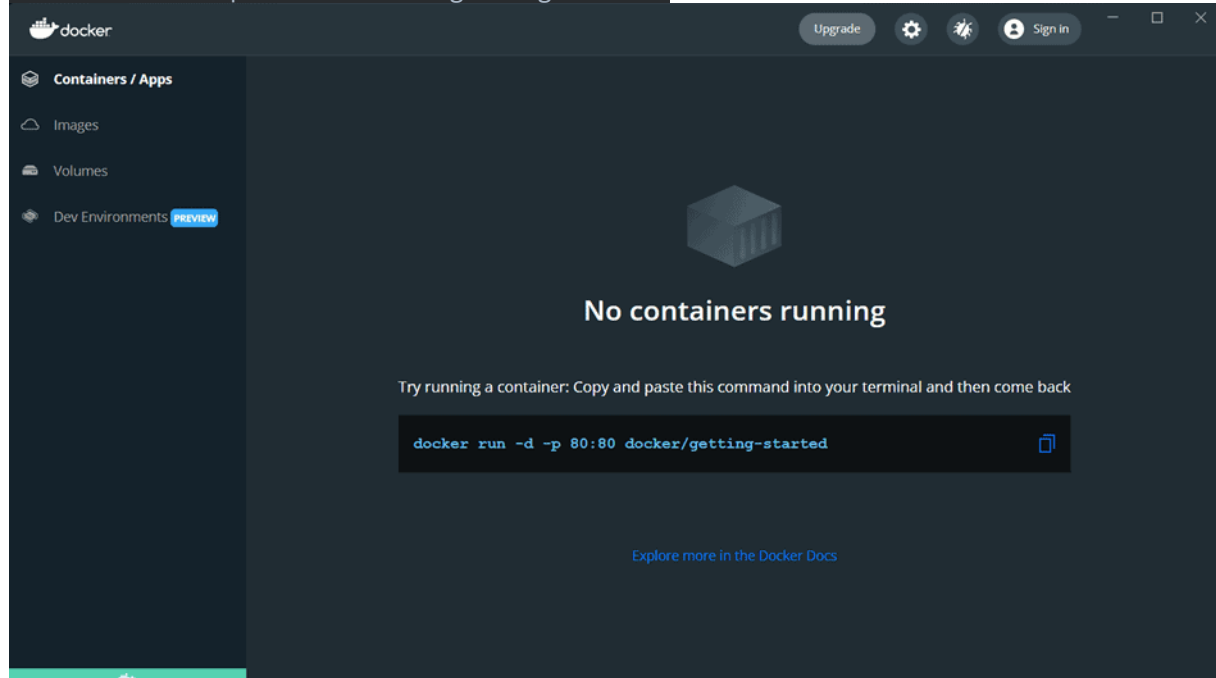
```
sudo apt-get install docker-ce
```

Lancer votre premier conteneur

Dans le reste de ce tutoriel, nous allons utiliser Docker pour Windows qui est maintenant prêt à l'emploi.

Pour lancer votre premier conteneur, il vous suffit de lancer la commande que vous voyez sur la première fenêtre affichée sur Docker à savoir

```
docker run -d -p 80:80 docker/getting-started
```



Copiez-la sur une invite de commande et tapez sur Entrer :

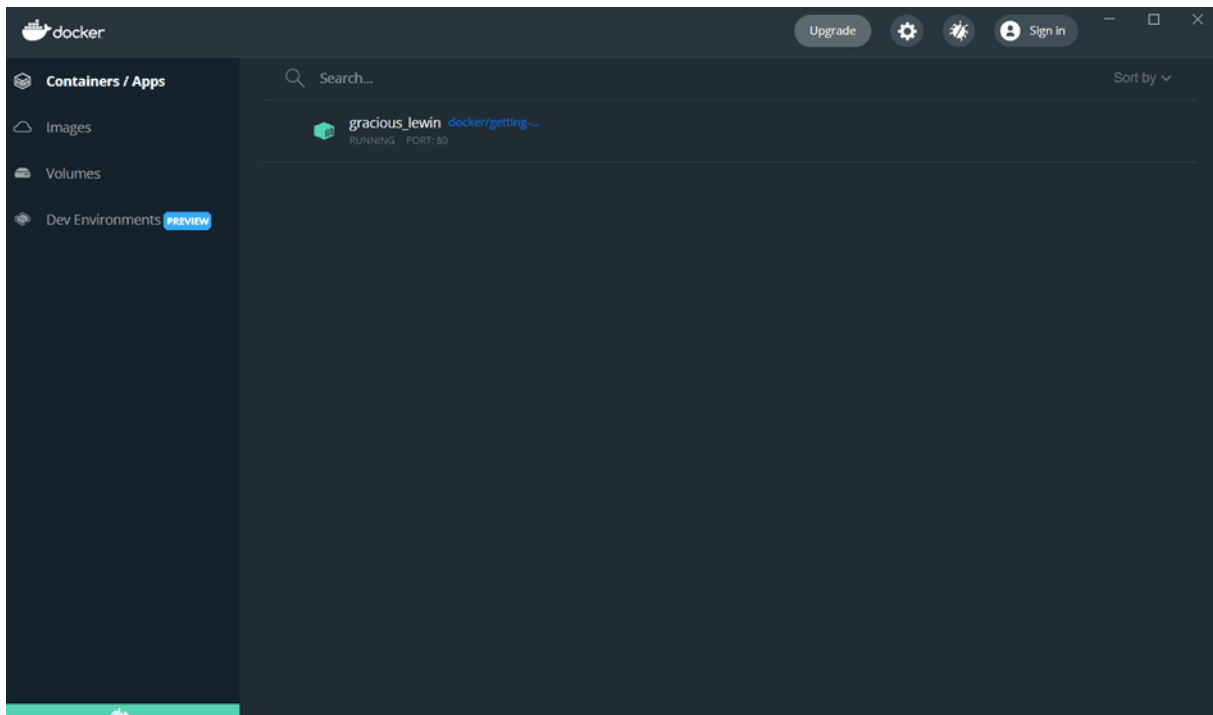
```
D:\Software\dev\cmdcr
λ docker run -d -p 80:80 docker/getting-started
Unable to find image 'docker/getting-started:latest' locally
latest: Pulling from docker/getting-started
59bfc3509f3: Downloading [====>] 276.9kB/2.818MB
8d6ba530f648: Downloading [=>] 232.8kB/7.341MB
5288d7ad7a7f: Download complete
39e51c61c033: Waiting
ee6f71c6f4a8: Waiting
f2303c6c8065: Waiting
0645fddcff40: Waiting
d05ee95f5d2f: Waiting
λ
```

C'est la commande run qui va exécuter l'image afin de créer le conteneur. Pour une première utilisation, Docker va tout d'abord télécharger l'image à partir de Docker Hub, comme l'image le montre. C'est-à-dire qu'il va effectuer un "**pull**". Dans le cas contraire, vous verrez l'ID de l'image apparaître une fois la commande exécutée.

```
C:\Docker\cmdcr
λ docker run -d -p 80:80 docker/getting-started
12ba0b288eae0cae06dce460cf915e50e89e68920f8dbf92690624dbe1984034

C:\Docker\cmdcr
λ
```

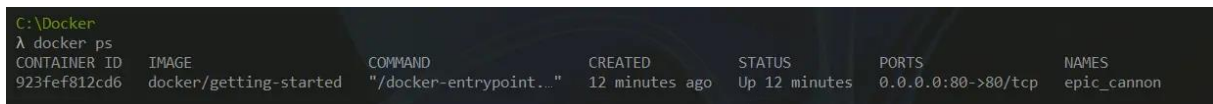
Vous pouvez ensuite vérifier sur votre fenêtre Docker que l'image s'exécute bien. Pour cela, il suffit de regarder l'interface de Docker, ce qui est le cas sur l'image ci-dessous :



Vous pouvez également le voir en tapant la ligne de commande suivante :

```
docker ps
```

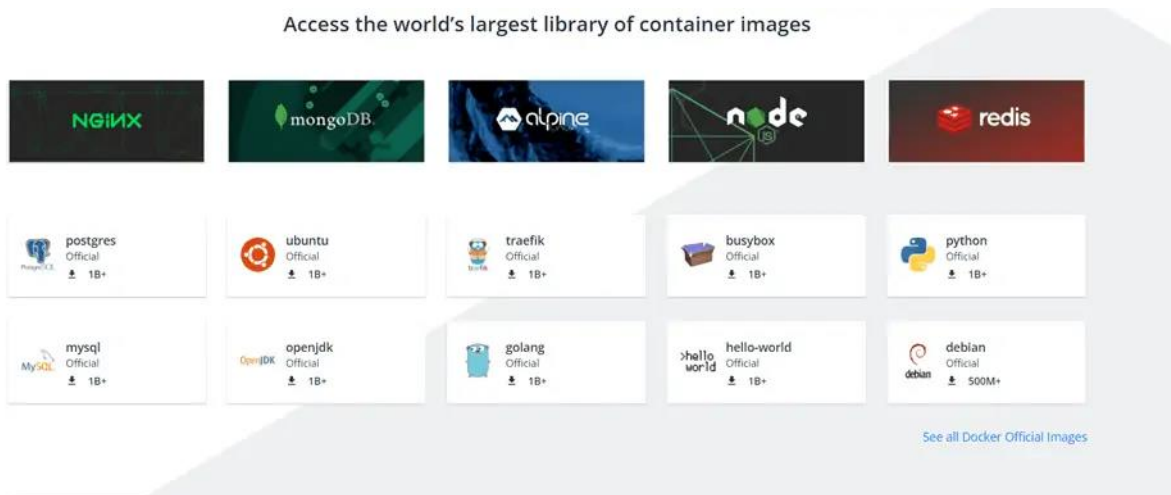
Cela vous renvoie la liste des conteneurs en cours d'exécution avec des détails les concernant :



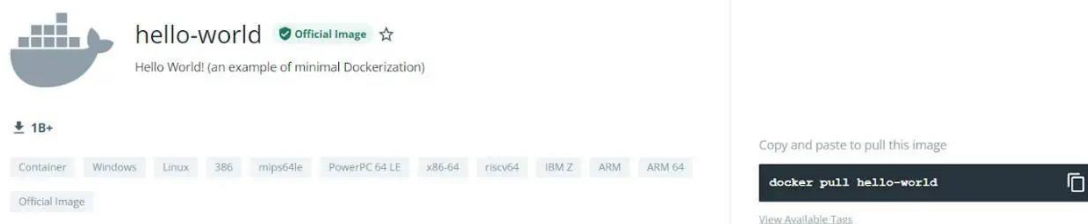
Bravo ! Vous avez lancé votre première image en quelques minutes.

Télécharger une image depuis le Docker Hub

Il faut savoir que dans la plupart des cas, vous allez vous baser sur des images existantes, y apporter quelques modifications afin de construire votre propre image. Il est donc important de savoir comment télécharger une image à partir de cette plateforme. Pour ce faire, rendez-vous sur Docker Hub et créez un compte.



Une fois cela fait, prenez l'image que vous souhaitez utiliser. Ici, nous allons prendre l'image *hello-world* de Docker Inc.



Vous l'aurez sans doute remarqué, mais il donne déjà la commande à exécuter pour le téléchargement, à savoir **docker pull hello-world**

Copiez cette commande sur votre invite de commande et appuyez sur *Entrer*.

```
C:\Docker
λ docker pull hello-world
Using default tag: latest
latest: Pulling from library/hello-world
2db29710123e: Pull complete
Digest: sha256:4c5f3db4f8a54eb1e017c385f683a2de6e06f75be442dc32698c9bbe6c861edd
Status: Downloaded newer image for hello-world:latest
docker.io/library/hello-world:latest
```

Une fois l'exécution terminée, vous pouvez consulter la liste des images présentes sur votre machine afin de vérifier si le téléchargement s'est bien passé. Pour cela, tapez la commande suivante :

```
docker images

C:\Docker
λ docker images
REPOSITORY          TAG         IMAGE ID      CREATED        SIZE
docker/getting-started latest      bd9a9f733898  4 weeks ago   28.8MB
hello-world          latest     feb5d9fea6a5  5 months ago  13.3kB
```

On peut voir ici que l'image *hello-world* a bien été téléchargée sur la machine.

Tutoriel Docker – Partie 1

Réalisation : Guillaume DELACROIX Formateur AFPA
30 août 2023

Afpa

Arrêter un conteneur

Pour arrêter un conteneur, il suffit de lancer la ligne de commande suivante :

```
docker stop id_du_conteneur
```

Pour connaître l'ID du conteneur, vous pouvez exécuter la commande qui liste tous les conteneurs en marche et récupérer la valeur de la première colonne ou le récupérer sur votre Docker Desktop (à privilégier car en listant via les lignes de commandes, les IDs sont « tronqués »).

Arrêtons par exemple le conteneur que l'on a lancé tout à l'heure.

```
C:\Docker
λ docker stop 923fef812cd6
923fef812cd6
```

Vous pouvez par la suite vérifier si le conteneur est bien stoppé de la même manière que lorsque vous l'avez lancé, à savoir en listant les conteneurs ou en vérifiant l'interface Docker.

Supprimer un conteneur

La commande qui sert à supprimer un conteneur est la suivante :

```
docker rm id_du_conteneur
```

```
C:\Docker
λ docker rm 923fef812cd6
923fef812cd6
```

À ce stade, vous n'aurez normalement plus de conteneurs sur votre machine. Mais si vous vérifiez dans la liste des images, vous verrez les deux images que nous avons utilisées jusqu'ici.

```
C:\Docker
λ docker images
REPOSITORY          TAG         IMAGE ID      CREATED        SIZE
docker/getting-started latest      bd9a9f733898 4 weeks ago    28.8MB
hello-world          latest     feb5d9fea6a5 5 months ago   13.3kB
```

Supprimer une image

Si vous souhaitez supprimer une image, vous devez taper cette commande :

```
docker rmi id_image
```

```
C:\Docker
λ docker rmi feb5d9fea6a5
Untagged: hello-world:latest
Untagged: hello-world@sha256:4c5f3db4f8a54eb1e017c385f683a2de6e06f75be442dc32698c9bbe6c861edd
Deleted: sha256:feb5d9fea6a5e9606aa995e879d862b825965ba48de054caab5ef356dc6b3412
Deleted: sha256:e07ee1baac5fae6a26f30cabfe54a36d3402f96afda318fe0a96cec4ca393359
```

Après l'exécution, si vous vérifiez la liste des images que vous possédez, vous n'aurez plus qu'une seule image disponible.

Créer un Dockerfile

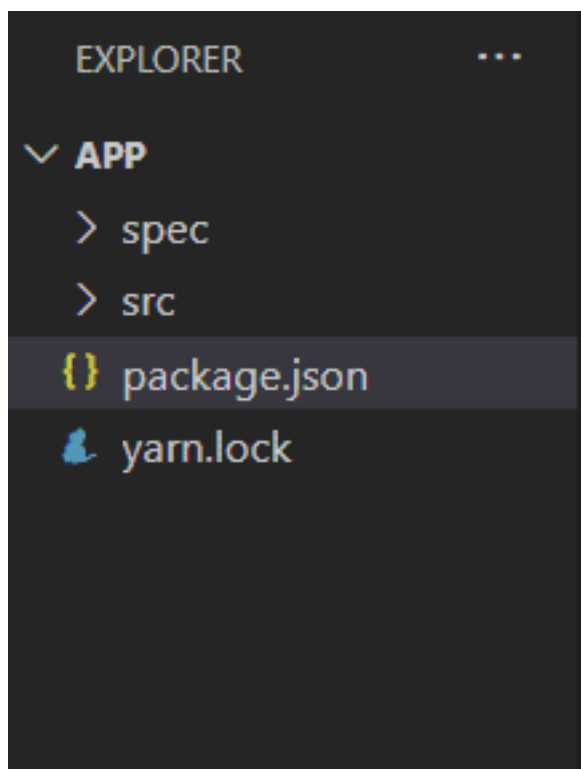
La création d'un Dockerfile vous permettra de créer votre propre image. Dans ce fichier, vous allez lister toutes les instructions dont vous aurez besoin. Cependant, veillez à ce que le nombre d'instructions soit le minimum possible, car, chaque instruction envoyée au serveur va créer une couche au-dessus de l'image que l'on appelle **Layer**. Plus le nombre de Layer est élevé, plus votre image sera lourde.

Voici les commandes que l'on retrouve fréquemment dans un Dockerfile :

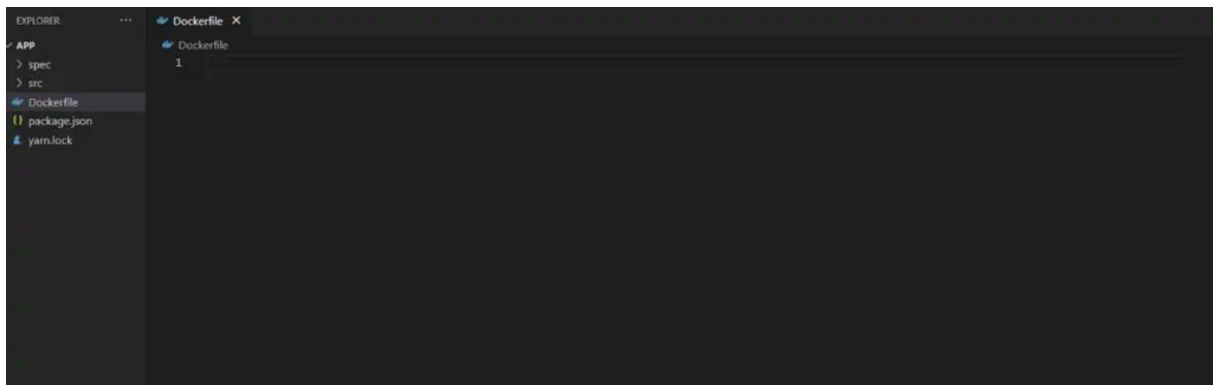
- **FROM** : c'est la commande qui sert à définir l'image de base de l'application ;
- **LABEL** : qui permet d'ajouter des métadonnées à l'image ;
- **RUN** : qui sert à exécuter des instructions dans le conteneur ;
- **ADD** : pour ajouter des fichiers dans l'image ;
- **WORKDIR** : afin de spécifier le répertoire dans lequel sera exécuté tout l'ensemble des instructions ;
- **EXPOSE** (facultatif) : qui sert à spécifier le port que l'on souhaite utiliser ;
- **VOLUME** (facultatif) : pour spécifier le répertoire à partager avec l'hôte ;
- **CMD** : qui sert à indiquer au conteneur la commande à exécuter lors de son lancement.

Pour illustrer tout cela, partons d'un projet que vous pouvez retrouver sur ce lien : <https://github.com/docker/getting-started/tree/master/app>

Une fois le projet téléchargé, extrayez le fichier .zip, copiez le dossier app à l'emplacement que vous souhaitez, puis ouvrez-le dans un éditeur de code.



Il faut maintenant créer le fichier Dockerfile au même niveau que le package.json en s'assurant qu'il n'y a pas d'extensions à celui-ci.



Copiez les instructions ci-dessous dans ce fichier afin de spécifier les caractéristiques de l'image que nous souhaitons :

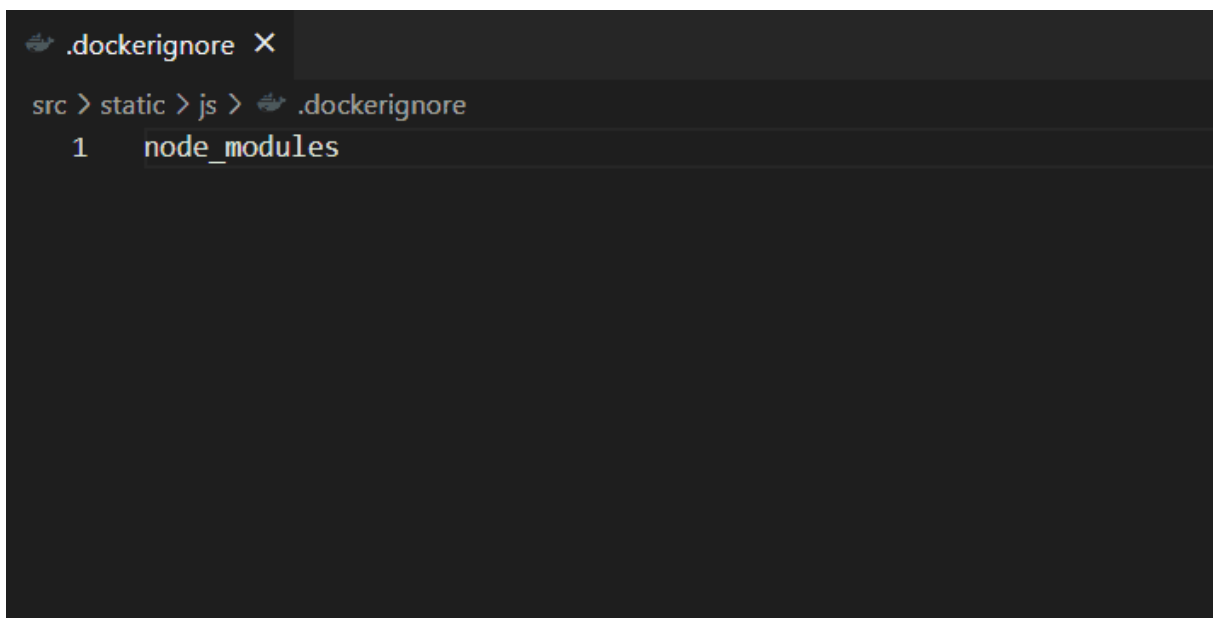
```
FROM node:14-alpine
RUN apk update && apk add --no-cache python3 g++ make

WORKDIR /app
COPY . .
RUN yarn install --production
CMD ["node", "src/index.js"]
EXPOSE 3000
```

Ici, nous avons spécifié l'image de base que nous allons utiliser qui est Node.js ainsi que la page à exécuter une fois le conteneur en marche.

Créer un fichier .dockerignore

Avant de pouvoir disposer de votre propre image, il faut également que le projet contienne un fichier .dockerignore pour ne pas copier certains fichiers lors de l'exécution du conteneur. Dans notre cas, voici ce que ce fichier doit contenir :



Ce fichier doit se situer au même niveau que le dockerfile.

Une fois que tout est en place, vous pouvez maintenant construire votre propre image. Pour cela, exécuter la commande suivante :

[illegible]

Une fois que l'exécution de la commande de construction se termine, vous pouvez vérifier si votre image est belle et bien disponible.

Pour exécuter le conteneur, la commande reste la même que ce que nous avons déjà vu un peu plus haut, mais cette fois-ci, changez le port à utiliser.

Une fois cette commande faite, rendez-vous sur votre navigateur et tapez le lien :

<http://localhost:3000/>. Vous allez voir que votre application s'affichera comme ceci :

No items yet! Add one above!

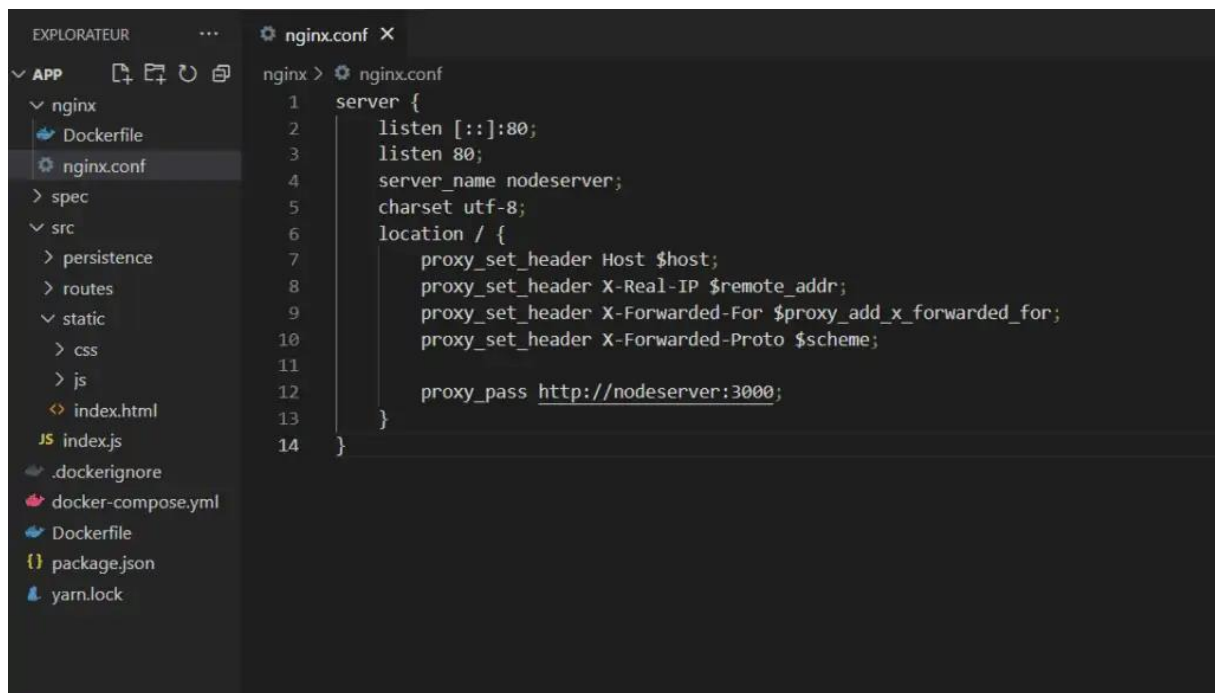
Lancer une application dans un serveur Nginx

Pour que le serveur Nginx puisse héberger correctement l'application, il faut que l'on effectue quelques modifications à l'application que nous venons de lancer dans le conteneur ci-dessus.

Tout d'abord, nous allons configurer notre propre serveur Nginx pour y intégrer notre application. L'objectif est de pouvoir lancer le conteneur du Nginx de base. Mais au lieu d'afficher la page d'accueil de celui-ci, nous allons le rediriger vers notre application.

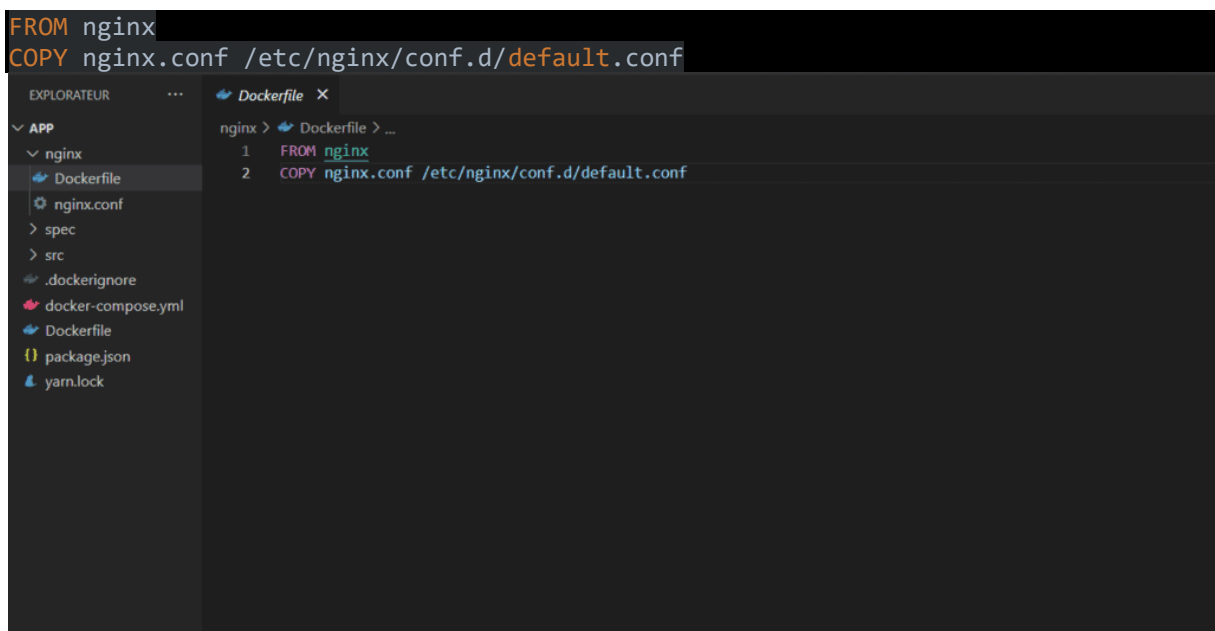
Pour cela, créez un dossier nommé "*nginx*" à la racine de notre projet. Ensuite, créez un fichier nommé "*nginx.conf*" dans lequel vous allez copier les instructions suivantes :

```
server {  
    listen [::]:80;  
    listen 80;  
    server_name nodeserver;  
    charset utf-8;  
    location / {  
        proxy_set_header Host $host;  
        proxy_set_header X-Real-IP $remote_addr;  
        proxy_set_header X-Forwarded-For $proxy_add_x_forwarded_for;  
        proxy_set_header X-Forwarded-Proto $scheme;  
  
        proxy_pass http://nodeserver:3000;  
    }  
}
```



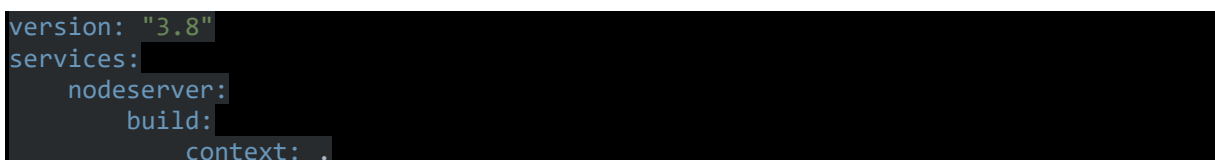
```
1 server {
2     listen [::]:80;
3     listen 80;
4     server_name nodeserver;
5     charset utf-8;
6     location / {
7         proxy_set_header Host $host;
8         proxy_set_header X-Real-IP $remote_addr;
9         proxy_set_header X-Forwarded-For $proxy_add_x_forwarded_for;
10        proxy_set_header X-Forwarded-Proto $scheme;
11
12        proxy_pass http://nodeserver:3000;
13    }
14 }
```

Toujours à l'intérieur de ce dossier, vous allez également créer un Dockerfile. Celui-ci contient la commande qui remplace *default.conf* (la configuration de base de Nginx) par celui que vous venez de créer. Voici les commandes que vous allez saisir à l'intérieur du fichier :



```
FROM nginx
COPY nginx.conf /etc/nginx/conf.d/default.conf
```

Maintenant, nous allons créer un docker-compose qui va lancer les deux conteneurs simultanément en référence aux deux Dockerfile du projet. Pour ce faire, créez un fichier appelé "*docker-compose.yml*" à la racine du projet puis ajoutez les instructions ci-dessous :



```
version: "3.8"
services:
  nodeserver:
    build:
      context: .
```

```
ports:
  - "3000:3000"
nginx:
  restart: always
  build:
    context: ./nginx
  ports:
    - "80:80"
```

EXPLORATEUR

- APP
 - nginx
 - Dockerfile
 - nginx.conf
 - spec
 - src
 - .dockerignore
 - docker-compose.yml
 - Dockerfile
 - package.json
 - yarn.lock

docker-compose.yml

```
1 version: "3.8"
2 services:
3   nodeserver:
4     build:
5       context: .
6     ports:
7       - "3000:3000"
8   nginx:
9     restart: always
10    build:
11      context: ./nginx
12    ports:
13      - "80:80"
```

Voilà à quoi ressemble un docker-compose.yml en général. À la première ligne, il faut spécifier la version utilisée. Ensuite, on spécifie les propriétés de chaque conteneur à lancer à l'intérieur de l'instruction « services ». Dans notre cas, *nodeserver* est l'application à lancer. Le contexte indique l'emplacement de cette dernière et le port sur lequel elle démarre.

Ensuite, on spécifie également les propriétés du serveur Nginx de la même manière.

Une fois que tout cela est fait, vous êtes prêt à lancer votre application dans un serveur Nginx.

Pour cela, tapez ce qui suit dans une invite de commande :

```
docker-compose up --build
```

```

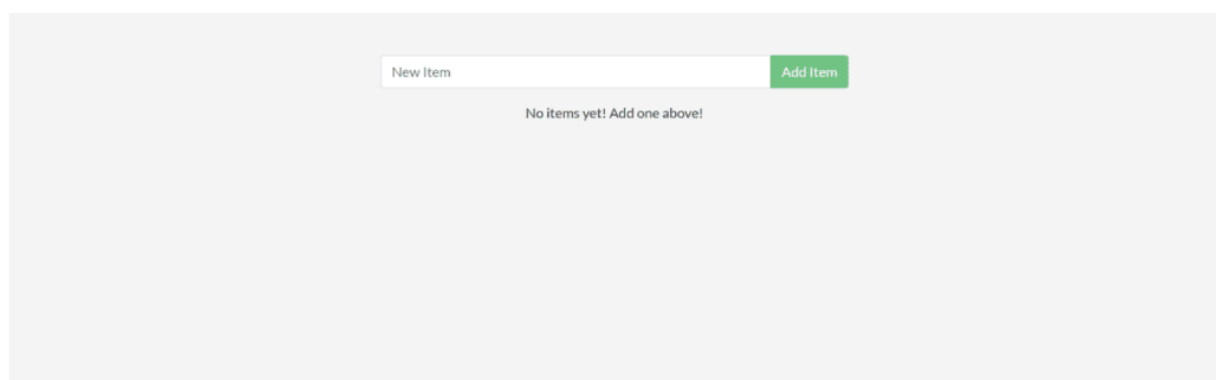
C:\Docker\app (101-app@1.0.0)
λ docker-compose up --build
Creating network "app_default" with the default driver
Building nodeserver
[+] Building 400.4s (10/10) FINISHED
=> [internal] load build definition from Dockerfile
=> => transferring Dockerfile: 12B
=> [internal] load .dockerignore
=> => transferring context: 1kB
=> [internal] load metadata for docker.io/library/node:12-alpine
=> [internal] load build context
=> => transferring context: 1.0kB
=> [1/5] FROM docker.io/library/node:12-alpine@sha256:543a0fcca3693a0ef5e7407eb70862be33c7c36cadd7424e77f5a8b64831041
=> CACHED [2/5] RUN apk add --no-cache python2 g++ make
=> CACHED [3/5] WORKDIR /app
=> [4/5] COPY ...
=> [5/5] RUN yarn install --production
=> exporting to image
=> => exporting layers
=> => writing image sha256:4c642be541de35b486f731ed1d78a1f10bd85c0be8086273ba18ed1398edc3
=> => naming to docker.io/library/app_nodeserver

Use 'docker scan' to run Snyk tests against images to find vulnerabilities and learn how to fix them
Building nginx
[+] Building 0.2s (7/7) FINISHED
=> [internal] load build definition from Dockerfile
=> => transferring Dockerfile: 11B
=> [internal] load .dockerignore
=> => transferring context: 1B
=> [internal] load metadata for docker.io/library/nginx:latest
=> [internal] load build context
=> => transferring context: 418B
=> CACHED [1/2] FROM docker.io/library/nginx
=> [2/2] COPY nginx.conf /etc/nginx/conf.d/default.conf
=> exporting to image
=> => exporting layers
=> => writing image sha256:a775ab4905bf08370a40e31fb1a4297f238a066a0b08a3df1a537013e7f4cc0
=> => naming to docker.io/library/app_nginx

Use 'docker scan' to run Snyk tests against images to find vulnerabilities and learn how to fix them
Creating app_nginx_1 ... done
Creating app_nodeserver_1 ... done
Attaching to app_nodeserver_1, app_nginx_1
nginx_1 | /docker-entrypoint.sh: /docker-entrypoint.d/ is not empty, will attempt to perform configuration
nginx_1 | /docker-entrypoint.sh: Looking for shell scripts in /docker-entrypoint.d/
nginx_1 | /docker-entrypoint.sh: Launching /docker-entrypoint.d/10-listen-on-ipv6-by-default.sh
nginx_1 | 10-listen-on-ipv6-by-default.sh: info: Getting the checksum of /etc/nginx/conf.d/default.conf

```

Dès que l'exécution se termine, rendez-vous sur votre navigateur et tapez localhost, l'adresse utilisée lorsque vous lancez un serveur Nginx de base. Vous allez constater que vous allez voir la même interface que nous avons lancée auparavant.



Partager une image

Deux possibilités s'offrent à vous si vous souhaitez partager votre image Docker à vos collaborateurs. La première est de le stocker sur un registre tel que Docker Hub. La seconde est d'exporter l'image et de le stocker dans l'emplacement que vous voulez. Voyons ces deux options une à une.

Partager sur Docker Hub

Voici les étapes à suivre afin d'héberger votre image sur Docker Hub

Tutoriel Docker – Partie 1

Réalisation : Guillaume DELACROIX Formateur AFPA
30 août 2023



- Connectez-vous sur votre compte Docker Hub à partir du site ;
- Cliquez sur “**Create Repository**”, spécifiez son nom (dans notre cas, ce sera mon-image) puis cliquez sur le bouton “**Create**” ;
- Reconnectez-vous sur votre compte Docker Hub, mais cette fois-ci, faites-le à partir de l’invite de commande en tapant ceci :

```
docker login -u votre-username
```

Note : remplacez “votre-username” par le nom d’utilisateur de votre compte Docker Hub.

- Reprenez l’image que nous avons créée auparavant, mais nous allons appliquer un tag sur le nom de l’image, comme ceci :

```
docker tag mon-image votre-username/mon-image
```

Cette commande vous permettra d’identifier l’image que vous allez partager en précisant votre nom d’utilisateur et le nom de l’image en question.

- Maintenant, vous allez pousser l’image sur Docker Hub en utilisant la commande suivante :

```
docker push votre-username/mon-image:latest
```

Voilà ! Votre image est disponible sur le référentiel que vous avez créé. Pour la récupérer, il faut lancer la commande **pull** que nous avons déjà vu un peu plus haut.

Exporter une image

Pour exporter une image, il vous suffit d’exécuter la commande suivante :

```
docker save mon-image:latest > mon-image.tar
```

Cette commande va renvoyer un fichier .tar qui va contenir tous les éléments qui composent l’image ainsi que leurs différentes couches.

Pour l’importer, la commande à utiliser sera :

```
docker load < my-image.tar
```

Nous terminons ce tutoriel et cet article ici. Nous avons traité beaucoup de facettes de Docker que vous pouvez désormais utiliser pour vos projets.

Si l’on résume, vous savez ce qu’est Docker, son histoire, ses avantages ainsi que les principaux composants de cette plateforme. Vous savez également, grâce au tutoriel, utiliser les images, les conteneurs, le serveur Nginx, le docker compose et enfin le partage d’images. Vous avez donc toutes les cartes en main pour bien débuter en Docker.