C’est quoi un DockerFile ? Comment utiliser et créer un DockerFile ? Comment exécuter un Dockerfile ?

Qui ne connaît pas Docker aujourd’hui ? Cette technologie de conteneurs offre de multiples avantages pour faire tourner des applications, y compris la flexibilité, l’efficacité, la portabilité, l’immutabilité, l’évolutivité et la sécurité. Après l’installation de Docker sur votre système, vous pouvez télécharger rapidement une image Docker depuis la registry publique ([**hub.docker.com**](http://hub.docker.com/)) pour un logiciel tel que WordPress. Ensuite, vous pouvez générer tout aussi rapidement un conteneur à partir de cette image sur votre machine pour faire tourner ce logiciel.

C’est déjà énorme ! Mais Docker peut vous emmener encore plus loin en vous permettant de créer, toujours facilement, vos propres images qui incorporent vos propres applications et les librairies propres à chaque application.

Pour ce faire, vous pourriez télécharger une image, créer un conteneur à partir de cette image, modifier le conteneur (par exemple, y installer votre application), puis utiliser la commande docker commit pour créer une image à partir du conteneur modifié.

Mais vous pouvez aussi utiliser Dockerfile pour construire d’emblée l’image Docker qu’il vous faut, couche par couche et de façon logique et organisée. Un Dockerfile est un fichier texte décrivant les différentes étapes permettant de partir d'une base pour aller vers une application fonctionnelle. Docker lit les instructions que vous mettez dans le Dockerfile pour créer automatiquement l’image requise.

En partant d’une image de base indiquée par la première instruction (FROM) dans un Dockerfile, Docker exécute chaque instruction dans l’ordre spécifié dans le Dockerfile et rajoute une couche supplémentaire par instruction. Si une instruction doit être modifiée à l’avenir, seulement les couches héritant de la couche de cette instruction seront affectées. Chaque couche est signée par un checksum SHA qui permet de détecter la moindre variation de contenu de la couche.

Ainsi, en mettant à la fin du Dockerfile les instructions pour les parties les plus susceptibles de changer, vous minimiser l’impact sur les autres couches. Vous éviterez également des répercussions en limitant les dépendances qui pourraient changer.

**Les bases sur DockerFile**

Les étapes principales pour utiliser Dockerfile sont :

1. Créer un fichier texte que vous nommerez « Dockerfile » sans extension. Il est possible de choisir d’autres noms pour ce fichier, mais Docker recherche en premier lieu un fichier qui s’appelle « Dockerfile ». Inutile de compliquer les choses pour le moment.
2. Mettez vos instructions dans ce fichier, par exemple, en vous servant d’un éditeur de texte standard comme Vi ou Vim ou un environnement de développement (IDE) approprié tel que VSCode qui permet avec les extensions adéquates de vérifier les bonnes pratiques Docker au passage.
3. Créer une image à partir de ce fichier via docker build.
4. Exécuter l’image via docker run pour générer un conteneur (docker run -it pour le mode interactif ou docker run -d pour le mode daemon).

Commençons par un exemple simple, à base de quatre commandes qui figurent dans tous ou dans la plupart des Dockerfiles :

FROM :  Il s’agit de la première instruction. Elle est indispensable, car elle définit l’image de base avec le système d’exploitation et l'application portant le reste des couches. Vous pouvez également spécifier une image de base minimale en utilisant l’instruction FROM scratch.  
COPY : Cette commande ajoute dans votre image des fichiers locaux ou distants tels le code source d’une application ou un fichier de configuration. Il existe aussi une commande ADD qui peut être très utile lorsque l'on veut extraire à la volée un tar.gz. ADD permet de faire ceci en une instruction alors que COPY ne permet que de copier la source vers une destination.  
RUN : Exécuter avec RUN toute commande souhaitée pour créer une nouvelle couche sur l’image.

CMD : Pour spécifier une commande à exécuter seulement lors du lancement du conteneur.  
Un Dockerfile simple pourrait alors ressembler à ceci :  
*FROM alpine:latest*  
*COPY . /app*  
*RUN apk add make && make /app*  
*CMD ["echo", "Prêt !"]*  
Dans cet exemple :

FROM crée une couche à partir de l’image Docker d’Alpine (une distribution de Linux très compacte et rapide à télécharger à partir de Docker Hub).

COPY rajoute des fichiers du répertoire dans le dossier /app.

RUN installe le binaire « make » sous Alpine et construit votre application avec « make ».

CMD se charge lors du lancement du conteneur d’afficher « Prêt ! » à l’écran.

Chaque instruction crée une couche sous forme de conteneur intermédiaire. Donc, il y en a quatre dans cet exemple et Docker passera par ces couches de 1 à 4 dans l’ordre indiqué ci-dessus.

La prochaine étape sera la transformation de ce Dockerfile en image. À ce stade, Docker recherche un fichier qui s’appelle « .dockerignore ». Ce fichier fonctionne de la même manière qu'un fichier .gitignore. Il indique qu’il ne faut pas copier les fichiers qui sont listés dedans comme par exemple les dumps de bases de données (trop volumineux) ou les clés d'accès à vos comptes clouds (trop sensibles). Donc s’il y en a besoin, créez votre fichier .dockerignore maintenant et mettez-le au même niveau que le Dockerfile.

Pour transformer votre Dockerfile en une image Docker, vous utiliserez la commande docker build de la manière suivante :*docker build -t nom\_image:nom\_tag localisation\_du\_Dockerfile*

Par exemple :

*docker build -t exemple\_image1:1.0 .*

L’option -t permet de nommer l’image. Dans cet exemple, le Dockerfile est dans le répertoire courant, donc « . ».

La commande docker images permet voir l’image ainsi créée dans la liste de toutes les images avec son nom ( « exemple\_image1 » et son « Image ID », par exemple, « db45f2dc112b ».

*Pour lancer le conteneur, utilisez la commande docker run avec le nom de l’image concernée, par exemple :*

*docker run -it exemple\_image1*

**Commandes additionnelles dans votre Dockerfile**

Bien sûr, il y a d’autres commandes encore que vous pouvez utiliser dans votre Dockerfile. Voici quelques exemples :

EXPOSE informe Docker que le conteneur est joignable sur les ports réseau spécifiés lors de l'exécution.

ENV définit une variable d'environnement.

ENTRYPOINT spécifie un binaire ou un script qui sera exécuté au runtime du conteneur.

VOLUME crée un point de montage entre le host et la couche dockerisée.

USER définit le nom d'utilisateur pour les commandes RUN / CMD / ENTRYPOINT qui suivent.

WORKDIR définit le répertoire de travail.

ARG définit une variable qui sera utilisée lors de la phase de build du container.

ONBUILD ajoute une instruction de déclenchement lorsque l'image est utilisée comme base pour une autre génération.

STOPSIGNAL définit le signal d'appel système qui sera envoyé au conteneur pour quitter.

LABEL applique des métadonnées clé/valeur à vos images, conteneurs ou démons.

Pour des informations plus complètes, vous pouvez consulter cette page (en anglais) : [**https://docs.docker.com/engine/reference/builder/**](https://docs.docker.com/engine/reference/builder/)

**Et maintenant pour le déploiement en ligne de votre Dockerfile ?**

Une fois que vous avez finalisé votre fichier DockerFile et que votre image correspond à vos attentes, vous avez qu’une seule envie, c’est de mettre en ligne votre application pour la partager, pas vrai ?

Cependant, **pour ce faire vous devez implémenter** : Une CI/CD avec des workers, mettre en place une registry privée pour stocker vos images, créer une plateforme Kubernetes sur un Cloud Provider, configurer un *load balancer* et les serveurs, ajouter un domaine, etc ….

**Il y a beaucoup plus simple** ! Il suffit **juste de pousser votre code chez Artifakt** incluant votre DockerFile afin de mettre en ligne votre application personnalisée.

**Comment cela fonctionne** ? C’est très simple, nous avons automatisé tout le [**processus de déploiement**](https://www.artifakt.io/fr/products/deploy) et création d’infrastructure. Nous vérifions la présence d’un DockerFile dans votre code et s’occupons de construire votre image comme vous le faites en local puis nous la poussons dans une registry privée afin de pouvoir déployer cette image sur notre infrastructure.

Enfin, nous vous fournissons une URL correspondant à votre environnement, cette URL pointe sur votre application (image). Votre application est en ligne ! Tout cela est **entièrement automatisé et géré** par notre technologie, vous pouvez donc vous concentrer sur votre code ou améliorer votre image et repousser le tout chez [**Artifakt**](https://www.artifakt.io/fr/) pour mettre à jour votre environnement.

Nous pouvons même aller encore plus vite et **ne pas s'embêter sur la construction d’un DockerFile**, si vous utilisez la console Artifakt, vous pouvez sélectionner le type d’application et nous vous fournirons par défaut un code et un dockerfile correspondant à cette application (ex : symfony, wordpress, nuxtjs, …), nous créerons par la même occasion un premier environnement avec cette application.