| **TP: INSTALLATION ET CONFIGURATION DE DOCKER** |
| --- |

**Les objectifs :**

− Installer Docker sur une machine virtuelle ubuntu

− Démarrer une application avec un conteneur Docker

− Metter à jour l’application et l’image Docker

− Partager l’application sur DockerHub

− Gérer des applications multi-conteneurs

− Utiliser Docker Compose

**Les étapes d’installation de Docker**

Avant installation de Docker Engine pour la première fois, il faut tout d’abord installer le dépôt Docker :

1. Installation de Docker repository:

| # Add Docker's official GPG key:  sudo apt-get update  sudo apt-get install ca-certificates curl gnupg  sudo install -m 0755 -d /etc/apt/keyrings  curl -fsSL https://download.docker.com/linux/ubuntu/gpg | sudo gpg -- dearmor -o /etc/apt/keyrings/docker.gpg  sudo chmod a+r /etc/apt/keyrings/docker.gpg  # Add the repository to Apt sources:  echo \  "deb [arch="$(dpkg --print-architecture)" signed  by=/etc/apt/keyrings/docker.gpg] https://download.docker.com/linux/ubuntu \ "$(. /etc/os-release && echo "$VERSION\_CODENAME")" stable" | \ sudo tee /etc/apt/sources.list.d/docker.list > /dev/null  sudo apt-get update |
| --- |

1

2. Installation des packages Docker.

Pour installer la dernière version de Docker:

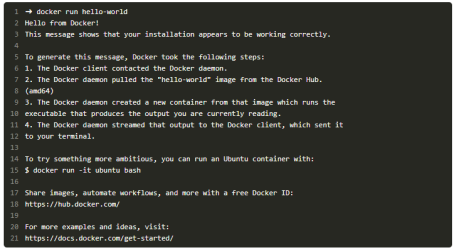
| sudo apt-get install docker-ce docker-ce-cli containerd.io docker-buildx plugin docker-compose-plugin |
| --- |

3. On va vérifier l’installation de Docker Engine en exécutant l’image

| hello-world : |
| --- |

| sudo docker run hello-world |
| --- |

Cette commande va télécharger une image de test et l’exécuter dans un container. Une fois le container exécuté, un message de confirmation sera affiché :



Dans ce cas, le daemon Docker va chercher si l'image hello-world est disponible en local. Dans le cas contraire, il va la récupérer sur la registry Docker officielle.

Le conteneur démarre, puis affiche du contenu, et il fini par s'arrêter. Si on souhaite que le conteneur reste allumé jusqu’à l'arrêt du service qu'il contient, on doit ajouter l’argument -- detach (-d) . Celui-ci permet de ne pas rester attaché au conteneur, et donc de pouvoir lancer plusieurs conteneurs.

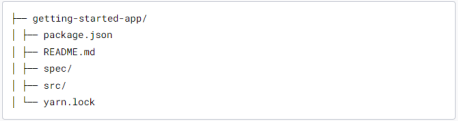
2

**Démarrez une application avec un conteneur Docker**

Nous commençons par récupérer le code source d’une application simple depuis github :

| git clone https://github.com/docker/getting-started-app.git |
| --- |

Le contenu du répertoire cloné devra être comme suit :



**Build une image de notre application :**

Pour créer l'image, on utilise un Dockerfile. Un Dockerfile est simplement un fichier texte sans extension qui contient un script d'instructions. Docker utilise ce script pour créer une image de conteneur.

1. Dans le répertoire getting-started-app, au même emplacement que le fichier package.json, créez un fichier nommé Dockerfile, dont le contenu est le suivant :

| # syntax=docker/dockerfile:1  FROM node:18-alpine  WORKDIR /app  COPY . .  RUN yarn install --production  CMD ["node", "src/index.js"]  EXPOSE 3000 |
| --- |

2. Builder l’image avec la commande suivante :

| docker build -t getting-started . |
| --- |

La commande docker build utilise le Dockerfile pour créer une nouvelle image. Docker télécharge beaucoup de « couches ». En effet, nous avons indiqué au constructeur qu’on souhaite démarrer à partir de l'image node:18-alpine, que Docker va télécharger.

3

Une fois que Docker a téléchargé l'image, les instructions du Dockerfile ont été copiées dans l’application et ont utilisé par yarn pour installer les dépendances de l’application. La directive CMD spécifie la commande par défaut à exécuter lors du démarrage d'un conteneur à partir de cette image.

L'argument -t permet de donner un nom à notre image Docker. Cela permet de retrouver plus facilement l’mage par la suite.

Le . est le répertoire où se trouve le Dockerfile ; dans notre cas, à la racine de notre projet.

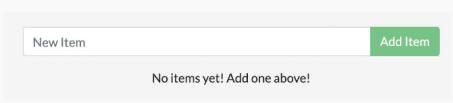
**Démarrer le container de votre application :**

Maintenant que vous disposez d'une image, vous pouvez exécuter l'application dans un conteneur à l'aide de la commande docker run.

1. Exécutez votre conteneur à l'aide de la commande docker run et spécifiez le nom de l'image que vous venez de créer :

| docker run -dp 127.0.0.1:3000:3000 getting-started |
| --- |

L'indicateur -d (abréviation de --detach) exécute le conteneur en arrière-plan. L'indicateur - p (abréviation de --publish) crée un mappage de port entre l'hôte et le conteneur. L'indicateur -p prend une valeur de chaîne au format HOST:CONTAINER, où HOST est l'adresse de l'hôte et CONTAINER est le port du conteneur. La commande publie le port 3000 du conteneur sur 127.0.0.1:3000 (localhost:3000) sur l'hôte. Sans le mappage des ports, vous ne pourriez pas accéder à l'application depuis l'hôte.

2. On peut voir l’application au niveau du navigateur Web sur http://localhost:3000: 

3. Ajoutez un élément ou deux et voyez qu'il fonctionne comme prévu. Vous pouvez marquer des éléments comme terminés et les supprimer. Votre frontend stocke avec succès les éléments dans le backend.

4

À ce stade, vous disposez d’un gestionnaire de liste de tâches en cours d’exécution avec quelques éléments.

4. On peut visualiser au moins un conteneur en cours d'exécution qui utilise l'image de démarrage et sur le port 3000.

La commande docker ps (ou docker ps -a) permet de visualiser tous les conteneurs en cours d’exécution :

| docker ps |
| --- |

Un résultat similaire à celui-ci devrait apparaître.

**Mise à jour de l’application :**

Nous avons conteneurisé une application de tâches. Dans cette partie, on va mettre à jour l'application et l'image. On verra comment arrêter et supprimer un conteneur.

**Mise à jour du code source de l’application :**

Dans les étapes suivantes, vous allez remplacer le « texte vide » lorsque vous n'avez aucun élément de liste de tâches par « Vous n'avez pas encore d'éléments de tâche ! Ajoutez-en un ci-dessus ! »

1. Dans le fichier src/static/js/app.js, mettez à jour la ligne 56 pour utiliser le nouveau texte vide.

| - <p className="text-center">No items yet! Add one above!</p> + <p className="text-center">You have no todo items yet! Add one above!</p> |
| --- |

2. Créez votre version mise à jour de l'image à l'aide de la commande docker build.

| docker build -t getting-started . |
| --- |

3. Démarrez un nouveau conteneur en utilisant le code mis à jour.

| docker run -dp 127.0.0.1:3000:3000 getting-started |
| --- |

5

Un message d’erreur s’affiche. L'erreur s'est produite car vous ne parvenez pas à démarrer le nouveau conteneur alors que votre ancien conteneur est toujours en cours d'exécution. La raison en est que l'ancien conteneur utilise déjà le port 3000 de l'hôte et qu'un seul processus sur la machine (conteneurs inclus) peut écouter un port spécifique. Pour résoudre ce problème, vous devez supprimer l'ancien conteneur.

**Supprimer l’ancien container:**

Pour supprimer un conteneur, vous devez d’abord l’arrêter. Une fois arrêté, vous pouvez le supprimer.

1. Obtenez l'ID du conteneur à l'aide de la commande docker ps.

| docker ps |
| --- |

2. Utilisez la commande docker stop pour arrêter le conteneur. Remplacez <the container-id> par l'ID de docker ps.

| docker stop <the-container-id> |
| --- |

3. Une fois le conteneur arrêté, vous pouvez le supprimer à l'aide de la commande docker rm.

| docker rm <the-container-id> |
| --- |

| ***Remarque:***  Vous pouvez arrêter et supprimer un conteneur en une seule commande en ajoutant l'indicateur de force à la commande docker rm. Par exemple : docker rm -f <the container-id> |
| --- |

**Démarrez le conteneur d'applications mis à jour**

Maintenant, démarrez votre application mise à jour à l’aide de la commande docker run.

| docker run -dp 127.0.0.1:3000:3000 getting-started |
| --- |

Actualisez votre navigateur sur http://localhost:3000 et vous devriez voir votre texte d'aide mis à jour.

**Partager l’application**

Maintenant que vous avez créé une image, vous pouvez la partager. Pour partager des images Docker, vous devez utiliser un registre Docker. Le registre par défaut est Docker Hub et c'est là que proviennent toutes les images que vous avez utilisées.

6

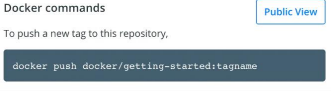
Pour pousher une image, vous devez d'abord créer un référentiel (repository) sur Docker Hub.

1. Inscrivez-vous ou connectez-vous à Docker Hub.

2. Sélectionnez le bouton Créer un référentiel.

3. Pour le nom du référentiel, utilisez Getting-started. Assurez-vous que la visibilité est publique.

4. Sélectionnez « créer »



Dans la ligne de commande, exécutez la commande docker push que vous voyez sur Docker Hub. Notez que votre commande aura votre identifiant Docker, et non « docker ».

| docker push docker/getting-started  The push refers to repository [docker.io/docker/getting-started] An image does not exist locally with the tag: docker/getting-started |
| --- |

Pourquoi a-t-il échoué ? La commande push recherchait une image nommée docker/getting started, mais n'en a pas trouvé. Si vous exécutez docker image ls, vous n'en verrez pas non plus.

Pour résoudre ce problème, vous devez « tager » votre image existante que vous avez créée pour lui donner un autre nom.

Connectez-vous à Docker Hub à l'aide de la commande docker login -u YOUR-USER NAME.

Utilisez la commande docker tag pour donner un nouveau nom à l'image de démarrage. Remplacez YOUR-USER-NAME par votre Docker ID.

| docker tag getting-started YOUR-USER-NAME/getting-started |
| --- |

Maintenant, exécutez à nouveau la commande docker push. Si vous copiez la valeur depuis Docker Hub, vous pouvez supprimer la partie tag name, car vous n'avez pas ajouté de balise

7

au nom de l'image. Si vous ne spécifiez pas de balise, Docker utilise une balise appelée latest.

| docker push YOUR-USER-NAME/getting-started |
| --- |

**Récupérez une image du Docker Hub**

On peut **récupérer des images sur le Docker Hub** sans pour autant lancer de conteneur. Pour cela, on a besoin de lancer la commande docker pull:

| docker pull ubuntu:22.04  22.04: Pulling from library/ubuntu  125a6e411906: Pull complete  Digest:  sha256:26c68657ccce2cb0a31b330cb0be2b5e108d467f641c62e13ab40cbec258c68d Status: Downloaded newer image for ubuntu:22.04  docker.io/library/ubuntu:22.04 |
| --- |

**Applications multi-conteneurs**

****

**Container networking**

Les conteneurs, par défaut, s'exécutent de manière isolée et ne connaissent rien des autres processus ou conteneurs sur la même machine. Alors, pour pouvoir communiquer, il faut placer les deux conteneurs sur le même réseau.

**Démarrer MySQL :**

Il existe deux manières de mettre un conteneur sur un réseau :

• Attribuez le réseau lors du démarrage du conteneur.

• Connectez un conteneur déjà en cours d'exécution à un réseau.

8

Dans les étapes suivantes, vous allez d'abord créer le réseau, puis attacher le conteneur MySQL au démarrage.

**Créer le réseau**

| docker network create todo-app |
| --- |

Démarrez un conteneur MySQL et attachez-le au réseau. Vous allez également définir quelques variables d'environnement que la base de données utilisera pour initialiser la base de données :

| docker run -d \  --network todo-app --network-alias mysql \  -v todo-mysql-data:/var/lib/mysql \  -e MYSQL\_ROOT\_PASSWORD=secret \  -e MYSQL\_DATABASE=todos \  mysql:8.0 |
| --- |

Pour confirmer que la base de données est opérationnelle, connectez-vous à la base de données et vérifiez qu'elle se connecte.

| docker exec -it <mysql-container-id> mysql -u root -p |
| --- |

Lorsque l’invite du mot de passe apparaît, saisissez secret. Dans le shell MySQL, répertoriez les bases de données et vérifiez que vous voyez la base de données todos.

| mysql> SHOW DATABASES; |
| --- |

Vous devriez voir un résultat ressemblant à ceci :



Quittez le shell MySQL pour revenir au shell sur votre machine.

| mysql> exit |
| --- |

9

Vous disposez maintenant d'une base de données todo et elle est prête à être utilisée.

**Se connecter à MySQL :**

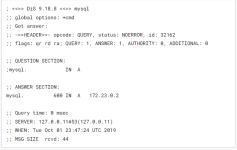
Démarrez un nouveau conteneur en utilisant l'image nicolaka/netshoot. Assurez-vous de le connecter au même réseau.

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

À l'intérieur du conteneur, vous allez utiliser la commande dig, qui est un outil DNS utile. Vous allez rechercher l'adresse IP du nom d'hôte mysql.

| dig mysql |
| --- |

Vous devriez obtenir une sortie comme celle-ci.



Dans la "SECTION RÉPONSE", vous verrez un enregistrement A pour MySQL qui se résout en 172.23.0.2 (votre adresse IP aura très probablement une valeur différente). Bien que mysql ne soit normalement pas un nom d'hôte valide, Docker a pu le résoudre en l'adresse IP du conteneur qui possédait cet alias réseau. N'oubliez pas que vous avez utilisé --network-alias plus tôt.

Cela signifie que votre application doit simplement se connecter à un hôte nommé mysql et communiquer avec la base de données.

**Exécutez votre application avec MySQL**

L'application todo prend en charge la définition de quelques variables d'environnement pour spécifier les paramètres de connexion MySQL. Ils sont:

• MYSQL\_HOST - le nom d'hôte du serveur MySQL en cours d'exécution

10

• MYSQL\_USER - le nom d'utilisateur à utiliser pour la connexion

• MYSQL\_PASSWORD - le mot de passe à utiliser pour la connexion

• MYSQL\_DB - la base de données à utiliser une fois connecté

Vous pouvez maintenant démarrer votre conteneur prêt pour le développement.

1. Spécifiez chacune des variables d'environnement précédentes et connectez le conteneur à votre réseau d'applications. Assurez-vous que vous vous trouvez dans le répertoire getting-started-app lorsque vous exécutez cette commande.

| docker run -dp 127.0.0.1:3000:3000 \  -w /app -v "$(pwd):/app" \  --network todo-app \  -e MYSQL\_HOST=mysql \  -e MYSQL\_USER=root \  -e MYSQL\_PASSWORD=secret \  -e MYSQL\_DB=todos \  node:18-alpine \  sh -c "yarn install && yarn run dev" |
| --- |

2. Si vous consultez les journaux du conteneur (docker logs -f <container-id>), vous devriez voir un message similaire au suivant, qui indique qu'il utilise la base de données MySQL.



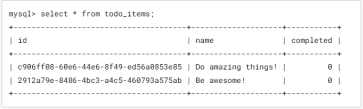
3. Ouvrez l'application dans votre navigateur et ajoutez quelques éléments à votre liste de tâches.

4. Connectez-vous à la base de données MySQL et prouvez que les éléments sont en cours d'écriture dans la base de données.

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

Et dans le shell mysql, exécutez ce qui suit :

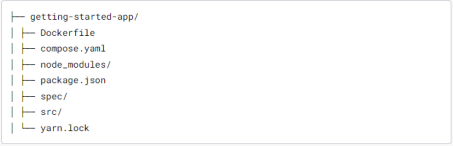
11

**Utiliser Docker Composer**

Docker Compose est un outil qui vous aide à définir et partager des applications multi conteneurs. Avec Compose, vous pouvez créer un fichier YAML pour définir les services et avec une seule commande, vous pouvez tout faire tourner ou tout détruire.

Le gros avantage de Compose est que vous pouvez définir votre pile d'applications dans un fichier, la conserver à la racine du référentiel de votre projet (sa version est désormais contrôlée) et permettre facilement à quelqu'un d'autre de contribuer à votre projet. Il suffirait à quelqu'un de cloner votre référentiel et de démarrer l'application à l'aide de Compose.

**Créer le fichier Composer**

Dans le répertoire getting-started-app, créez un fichier nommé compose.yaml. **Définir le service d'application**

Utilisez la commande suivante pour démarrer le service d'application.

docker run -dp 127.0.0.1:3000:3000 \

-w /app -v "$(pwd):/app" \

--network todo-app \

12

-e MYSQL\_HOST=mysql \

-e MYSQL\_USER=root \

-e MYSQL\_PASSWORD=secret \

-e MYSQL\_DB=todos \

node:18-alpine \

sh -c "yarn install && yarn run dev"

Vous allez maintenant définir ce service dans le fichier compose.yaml.

1. Ouvrez compose.yaml dans un éditeur de texte ou de code et commencez par définir le nom et l'image du premier service (ou conteneur) que vous souhaitez exécuter dans le cadre de votre application. Le nom deviendra automatiquement un alias réseau, ce qui sera utile lors de la définition de votre service MySQL.

| services:  app:  image: node:18-alpine |
| --- |

2. En règle générale, vous verrez commande proche de la définition image, ajoutez la commande à votre fichier compose.yaml.

| services:  app:  image: node:18-alpine  command: sh -c "yarn install && yarn run dev" |
| --- |

3. Migrez maintenant la partie -p 127.0.0.1:3000:3000 de la commande en définissant les ports du service.

| services:  app:  image: node:18-alpine  command: sh -c "yarn install && yarn run dev"  ports:  - 127.0.0.1:3000:3000 |
| --- |

4. Ensuite, migrez à la fois le répertoire de travail (-w /app) et le mappage de volume (-v "$(pwd):/app") en utilisant les définitions working\_dir et volumes.

13

L'un des avantages des définitions de volume Docker Compose est que vous pouvez utiliser des chemins relatifs à partir du répertoire actuel.

| services:  app:  image: node:18-alpine  command: sh -c "yarn install && yarn run dev"  ports:  - 127.0.0.1:3000:3000  working\_dir: /app  volumes:  - ./:/app |
| --- |

Enfin, vous devez migrer les définitions des variables d'environnement à l'aide de la clé environnement.

| services:  app:  image: node:18-alpine  command: sh -c "yarn install && yarn run dev"  ports:  - 127.0.0.1:3000:3000  working\_dir: /app  volumes:  - ./:/app  environment:  MYSQL\_HOST: mysql  MYSQL\_USER: root  MYSQL\_PASSWORD: secret  MYSQL\_DB: todos |
| --- |

**Définir le service MySQL**

Il est maintenant temps de définir le service MySQL

docker run -d \

--network todo-app --network-alias mysql \

-v todo-mysql-data:/var/lib/mysql \

-e MYSQL\_ROOT\_PASSWORD=secret \

14

-e MYSQL\_DATABASE=todos \

mysql:8.0

Définissez d’abord le nouveau service et nommez-le mysql afin qu’il obtienne automatiquement l’alias du réseau. Spécifiez également l’image à utiliser.

| services:  app:  # The app service definition  mysql:  image: mysql:8.0 |
| --- |

Ensuite, définissez le mappage de volume. Lorsque vous avez exécuté le conteneur avec Docker Run, Docker a créé automatiquement le volume nommé. Cependant, cela ne se produit pas lors de l'exécution avec Compose.

| services:  app:  # The app service definition  mysql:  image: mysql:8.0  volumes:  - todo-mysql-data:/var/lib/mysql  volumes:  todo-mysql-data: |
| --- |

Enfin, vous devez spécifier les variables d'environnement.

services:

app:

# The app service definition

mysql:

image: mysql:8.0

volumes:

- todo-mysql-data:/var/lib/mysql

environment:

MYSQL\_ROOT\_PASSWORD: secret

MYSQL\_DATABASE: todos

15

volumes:

todo-mysql-data:

À ce stade, votre compose.yaml complet devrait ressembler à ceci :

| services:  app:  image: node:18-alpine  command: sh -c "yarn install && yarn run dev"  ports:  - 127.0.0.1:3000:3000  working\_dir: /app  volumes:  - ./:/app  environment:  MYSQL\_HOST: mysql  MYSQL\_USER: root  MYSQL\_PASSWORD: secret  MYSQL\_DB: todos  mysql:  image: mysql:8.0  volumes:  - todo-mysql-data:/var/lib/mysql  environment:  MYSQL\_ROOT\_PASSWORD: secret  MYSQL\_DATABASE: todos  volumes:  todo-mysql-data: |
| --- |

**Exécuter la pile d'applications**

Maintenant que vous disposez de votre fichier compose.yaml, vous pouvez démarrer votre application.

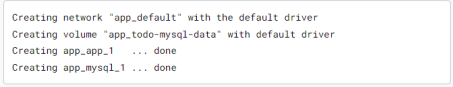
Assurez-vous qu'aucune autre copie des conteneurs n'est exécutée en premier. Utilisez docker ps pour lister les conteneurs et docker rm -f <ids> pour les supprimer.

16

Démarrez la pile d'applications à l'aide de la commande docker compose up. Ajoutez l'indicateur -d pour tout exécuter en arrière-plan.

| Docker-compose up -d |
| --- |

Lorsque vous exécutez la commande précédente, vous devriez voir un résultat semblable à celui-ci :



Vous remarquerez que Docker Compose a créé le volume ainsi qu'un réseau. Par défaut, Docker Compose crée automatiquement un réseau spécifiquement pour la pile d'applications (c'est pourquoi vous n'en avez pas défini dans le fichier Compose).

Consultez les journaux à l’aide de la commande docker-compose logs -f. Vous verrez les journaux de chacun des services entrelacés dans un seul flux. Ceci est utile lorsque vous souhaitez surveiller les problèmes liés au timing. L'indicateur -f suit le journal et vous donnera donc une sortie en direct au fur et à mesure de sa génération.

Si vous avez déjà exécuté la commande, vous verrez un résultat semblable à ceci :

| mysql\_1 | 2019-10-03T03:07:16.083639Z 0 [Note] mysqld: ready for connections.  mysql\_1 | Version: '8.0.31' socket: '/var/run/mysqld/mysqld.sock' port: 3306 MySQL Community Server (GPL)  app\_1 | Connected to mysql db at host mysql  app\_1 | Listening on port 3000 |
| --- |

Le nom du service est affiché au début de la ligne (souvent coloré) pour aider à distinguer les messages. Si vous souhaitez afficher les journaux d'un service spécifique, vous pouvez ajouter le nom du service à la fin de la commande logs (par exemple, docker compose logs -f app).

À ce stade, vous devriez pouvoir ouvrir votre application dans votre navigateur sur http://localhost:3000 et la voir fonctionner.

17

Lorsque vous êtes prêt à tout démolir, exécutez simplement Docker-Compose Down. Les conteneurs s'arrêteront et le réseau sera supprimé.

**Bonnes pratiques de création d’image**

**Superposition d'images**

À l’aide de la commande docker-image history, vous pouvez voir la commande qui a été utilisée pour créer chaque layer dans une image.

Utilisez la commande docker-image history pour voir les layers dans l'image de démarrage que vous avez créée.

| Docker image history getting-started |
| --- |

**Mise en cache des couches**

Le Dockerfile déjà crée est le suivant :

| # syntax=docker/dockerfile:1  FROM node:18-alpine  WORKDIR /app  COPY . .  RUN yarn install --production  CMD ["node", "src/index.js"] |
| --- |

En apportant n’importe quelle modification, toutes les dépendances sont à nouveau crée.

Pour résoudre ce problème, vous devez restructurer votre Dockerfile pour prendre en charge la mise en cache des dépendances. Pour les applications basées sur des nodes, ces dépendances sont définies dans le fichier package.json. Vous pouvez d'abord copier uniquement ce fichier, installer les dépendances, puis copier tout le reste. Ensuite, vous ne recréez les dépendances que s'il y a eu une modification dans le fichier package.json.

1. Mettez à jour le Dockerfile pour copier d'abord dans le package.json, installez les dépendances, puis copiez tout le reste.

# syntax=docker/dockerfile:1

FROM node:18-alpine

WORKDIR /app

18

COPY package.json yarn.lock ./

RUN yarn install --production

COPY . .

CMD ["node", "src/index.js"]

2. Créez un fichier nommé .dockerignore dans le même dossier que le Dockerfile avec le contenu suivant.

node\_modules

Les fichiers .dockerignore sont un moyen simple de copier uniquement les fichiers pertinents pour l'image. Dans ce cas, le dossier node\_modules doit être omis lors de la deuxième étape COPY car sinon, il risquerait d'écraser les fichiers créés par la commande lors de l'étape RUN.

3. Créer une nouvelle image à l'aide de Docker build

| docker build -t getting-started . |
| --- |

4. Maintenant, apportez une modification au fichier src/static/index.html. Par exemple, remplacez le <titre> par « The Awesome Todo App ».

5. Créez maintenant l'image Docker en utilisant docker build -t getting-started .

Vous devriez remarquer que la construction a été beaucoup plus rapide. Et vous verrez que plusieurs étapes utilisent des couches précédemment mises en cache. Pousser et extraire cette image et ses mises à jour seront également beaucoup plus rapides.

19