// Didacticiel //

Comment installer et utiliser Docker sur Ubuntu 20.04

Publié le 11 juin 2020

Docker Ubuntu 20.04

Par Brian Hogan
Développeur et auteur chez DigitalOcean.

English

Introduction

Docker est une application qui simplifie le processus de gestion des processus d'application dans les *conteneurs* . Les conteneurs vous permettent d'exécuter vos applications dans des processus isolés des ressources. Ils sont similaires aux machines virtuelles, mais les conteneurs sont plus portables, plus respectueux des ressources et plus dépendants du système d'exploitation hôte.

Pour une introduction détaillée aux différents composants d'un conteneur Docker, consultez l'Écosystème Docker : Une introduction aux composants communs .

Dans ce tutoriel, vous allez installer et utiliser Docker Community Edition (CE) sur Ubuntu 20.04. Vous allez installer Docker lui-même, travailler avec des conteneurs et des images, et pousser une image vers un référentiel Docker.

Conditions préalables

Pour suivre ce tutoriel, vous aurez besoin des éléments suivants :

- Un serveur Ubuntu 20.04 configuré en suivant le guide de configuration initiale de serveur Ubuntu 20.04 , comprenant un utilisateur non root avec privilèges sudo et un pare-feu.
- Un compte sur Docker Hub si vous souhaitez créer vos propres images et les pousser vers Docker Hub, comme indiqué dans les Étapes 7 et 8.

Étape 1 – Installation de Docker

Le package d'installation Docker disponible dans le référentiel officiel Ubuntu peut ne pas être la dernière version. Pour être sûr de disposer de la dernière version, nous allons installer Docker à partir du référentiel officiel Docker. Pour ce faire, nous allons ajouter une nouvelle source de paquets, ajouter la clé GPG de Docker pour nous assurer que les téléchargements sont valables, puis nous installerons le paquet.

Tout d'abord, mettez à jour votre liste de packages existants :

```
$ sudo apt update
```

Copie

Ensuite, installez quelques paquets pré-requis qui permettent `apt` d'utiliser les paquets sur HTTPS :

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

Ensuite, transférez la clé GPG du dépôt officiel de Docker à votre système :

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

Indiquez le référentiel Docker aux sources APT :

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

Ensuite, mettez à jour la base de données des paquets avec les paquets Docker à partir du référentiel qui vient d'être ajouté :

```
$ sudo apt update
```
Copie

vérifiez-vous que vous êtes sur le point d'installer à partir du dépôt Docker et non du dépôt Ubuntu par défaut :

```
$ apt-cache policy docker-ce
```
Copie

Vous verrez un résultat comme celui-ci, bien que le numéro de version du Docker puisse être différent :

Sortie de la politique apt-cache docker-ce

```
docker-ce:
  Installed: (none)
  Candidate: 5:19.03.9~3-0~ubuntu-focal
  Version table:
     5:19.03.9~3-0~ubuntu-focal 500
        500 https://download.docker.com/linux/ubuntu focal/stable amd64 Packages
```

Notez que le `docker-ce` n'est pas installé, mais que le candidat à l'installation provient du dépôt Docker pour Ubuntu 20.04 (`focal`).

Enfin, installez Docker :

```
$ sudo apt install docker-ce
```
Copie

Le Docker devrait maintenant être installé, le démonter, et le processus autorisé à démarrer au démarrage. Vérifiez qu'il tourne :

```
$ sudo systemctl status docker
```
Copie

La sortie devrait être similaire à ce qui suit, montrant que le service est actif et en cours d'exécution :

```
Output
● docker.service - Docker Application Container Engine
     Loaded: loaded (/lib/systemd/system/docker.service; enabled; vendor preset: enabled)
     Active: active (running) since Tue 2020-05-19 17:00:41 UTC; 17s ago
TriggeredBy: ● docker.socket
       Docs: https://docs.docker.com
   Main PID: 24321 (dockerd)
      Tasks: 8
     Memory: 46.4M
     CGroup: /system.slice/docker.service
             └─24321 /usr/bin/dockerd -H fd:// --containerd=/run/containerd/containerd.sock
```

L'installation de Docker vous donne maintenant non seulement le service Docker (démon) mais aussi l'utilitaire en ligne de commande `docker` , ou le client Docker. Nous allons voir comment utiliser la commande `docker` plus loin dans ce tutoriel.

Étape 2 – Exécution de la commande Docker sans sudo (facultatif)

Par défaut, la commande `docker` ne peut être exécutée que par l'utilisateur **root** ou par un utilisateur du groupe **docker** , qui est automatiquement créée lors du processus d'installation de Docker. Si vous essayez d'exécuter la commande `docker` sans la faire précéder de `sudo` ou sans être dans le groupe **docker** , vous échouerez un résultat comme celui-ci :

```
Output
docker: Cannot connect to the Docker daemon. Is the docker daemon running on this host?.
See 'docker run --help'.
```

Si vous voulez éviter de taper `sudo` chaque fois que vous exécutez la commande `docker` , composez votre nom d'utilisateur au groupe `docker` :

```
$ sudo usermod -aG docker ${USER}
```
Copie

Pour appliquer la nouvelle appartenance au groupe, déconnectez-vous du serveur et reconnectez-vous, ou saisi ce qui suit :

```
$ su - ${USER}
```
Copie

Vous serez invité à saisir le mot de passe utilisateur pour continuer.

Vérifiez que votre utilisateur est maintenant ajouté au groupe **docker** en tapant :

```
$ id -nG
```
Copie

```
Output
sammy sudo docker
```

Si vous devez ajouter un utilisateur au groupe `docker` pour lequel vous n'êtes pas connecté, déclarez ce nom d'utilisateur autorisé :

```
$ sudo usermod -aG docker username
```
Copie

La suite de cet article suppose que vous transposiez la commande `docker` en tant qu'utilisateur dans le groupe **docker** . Si vous choisissez de ne pas le faire, veuillez faire précéder les commandes de `sudo` .

Examinons maintenant la commande `docker` .

Étape 3 – Utilisation de la commande Docker

L'utilisation `docker` consiste à lui faire passer une chaîne suivie d'options et de commandes d'arguments. La syntaxe prend cette forme :

```
$ docker [option] [command] [arguments]
```
Copie

Pour voir toutes les sous-commandes disponibles, notées :

```
$ docker
```
Copie

À partir du docker 19, la liste complète des sous-commandes disponibles est incluse :

```
Output
  attach      Attach local standard input, output, and error streams to a running container
  build       Build an image from a Dockerfile
  commit      Create a new image from a container's changes
  cp          Copy files/folders between a container and the local filesystem
  create      Create a new container
  diff        Inspect changes to files or directories on a container's filesystem
  events      Get real time events from the server
  exec        Run a command in a running container
  export      Export a container's filesystem as a tar archive
  history     Show the history of an image
  images      List images
  import      Import the contents from a tarball to create a filesystem image
  info        Display system-wide information
```

```
  inspect     Return low-level information on Docker objects
  kill        Kill one or more running containers
  load        Load an image from a tar archive or STDIN
  login       Log in to a Docker registry
  logout      Log out from a Docker registry
  logs        Fetch the logs of a container
  pause       Pause all processes within one or more containers
  port        List port mappings or a specific mapping for the container
  ps          List containers
  pull        Pull an image or a repository from a registry
  push        Push an image or a repository to a registry
  rename      Rename a container
  restart     Restart one or more containers
  rm          Remove one or more containers
  rmi         Remove one or more images
  run         Run a command in a new container
  save        Save one or more images to a tar archive (streamed to STDOUT by default)
  search      Search the Docker Hub for images
  start       Start one or more stopped containers
  stats       Display a live stream of container(s) resource usage statistics
  stop        Stop one or more running containers
  tag         Create a tag TARGET_IMAGE that refers to SOURCE_IMAGE
  top         Display the running processes of a container
  unpause     Unpause all processes within one or more containers
  update      Update configuration of one or more containers
  version     Show the Docker version information
  wait        Block until one or more containers stop, then print their exit codes
```

Pour voir les options disponibles pour une commande spécifique, désignées :

```
$ docker docker-subcommand --help
```

Copie

Pour voir les informations sur Docker à l'échelle du système, utilisez :

```
$ docker info
```

Copie

Examens spécifiques de ces commandes. Nous allons commencer par travailler avec des images.

Étape 4 – Travailler avec des images Docker

Les conteneurs Docker sont intégrés à partir d'images Docker. Par défaut, Docker tire ces images de Docker Hub , un registre Docker géré par Docker, l'entreprise à l'origine du projet Docker. Tout le monde peut héberger ses images Docker sur Docker Hub, de sorte que la plupart des applications et des distributions Linux dont vous aurez besoin y auront des images hébergées.

Pour vérifier si vous pouvez accéder et télécharger des images de Docker Hub, notez :

```
$ docker run hello-world
```

Copie

La sortie indiquant que Docker fonctionne correctement :

```
Output
Unable to find image 'hello-world:latest' locally
latest: Pulling from library/hello-world
0e03bdcc26d7: Pull complete
Digest: sha256:6a65f928fb91fcfbc963f7aa6d57c8eeb426ad9a20c7ee045538ef34847f44f1
Status: Downloaded newer image for hello-world:latest

Hello from Docker!
This message shows that your installation appears to be working correctly.

...
```

Au départ, Docker n'a pas pu trouver l'image `hello-world` localement, il a donc téléchargé l'image depuis Docker Hub, qui est le référentiel par défaut. Une fois l'image téléchargée, Docker a créé un conteneur à partir de l'image et l'application dans le conteneur s'est exécutée, simplifié le message.

Vous pouvez rechercher des images disponibles sur Docker Hub en utilisant la commande `docker` avec la sous-commande `search` . Par exemple, pour rechercher l'image Ubuntu, notée :

```
$ docker search ubuntu
```
Copie

Le script va progresser Docker Hub et retourner une liste de toutes les images dont le nom correspond à la chaîne de recherche. Dans ce cas, la sortie sera similaire à celle-ci :

```
Output
NAME                                      DESCRIPTION
ubuntu                                    Ubuntu is a Debian-based Linux operating sys.
dorowu/ubuntu-desktop-lxde-vnc            Docker image to provide HTML5 VNC interface .
rastasheep/ubuntu-sshd                    Dockerized SSH service, built on top of offi.
consol/ubuntu-xfce-vnc                    Ubuntu container with "headless" VNC session.
ubuntu-upstart                            Upstart is an event-based replacement for th.
ansible/ubuntu14.04-ansible               Ubuntu 14.04 LTS with
...
```

Dans la colonne **OFFICIAL** , **OK** indique une image construite et soutenue par l'entreprise à l'origine du projet. Une fois que vous avez identifié l'image que vous souhaitez utiliser, vous pouvez la télécharger sur votre ordinateur à l'aide de la sous-commande `pull` .

Exécutez la commande suivante pour télécharger l'image officielle d' `ubuntu` sur votre ordinateur

```
$ docker pull ubuntu
```
Copie

Vous verrez la sortie suivante :

```
Output
Using default tag: latest
latest: Pulling from library/ubuntu
d51af753c3d3: Pull complete
fc878cd0a91c: Pull complete
6154df8ff988: Pull complete
fee5db0ff82f: Pull complete
Digest: sha256:747d2dbbaaee995098c9792d99bd333c6783ce56150d1b11e333bbceed5c54d7
Status: Downloaded newer image for ubuntu:latest
docker.io/library/ubuntu:latest
```

Une fois qu'une image a été téléchargée, vous pouvez alors lancer un conteneur en utilisant l'image téléchargée avec la sous-commande `run` . Comme vous l'avez vu avec l'exemple `hello-world` , si une image n'a pas été téléchargée `docker` lorsqu'elle est exécutée avec la sous-commande `run` , le client Docker téléchargera d'abord l'image, puis lancera un conteneur en l'utilisant.

Pour voir les images qui ont été téléchargées sur votre ordinateur, écrites :

```
$ docker images
```
Copie

La sortie utiliséea à ce qui suit :

```
Output
REPOSITORY          TAG                 IMAGE ID            CREATED             SIZE
ubuntu              latest              1d622ef86b13        3 weeks ago         73.9MB
hello-world         latest              bf756fb1ae65        4 months ago        13.3kB
```

Comme vous le révélez plus loin dans ce tutoriel, les images que vous utilisez pour gérer les conteneurs peuvent être modifiées et utilisées pour générer de nouvelles images, qui peuvent ensuite être importées (*poussées* est le terme technique) vers Docker Hub ou d'autres registres Docker.

Voyons commenté des conteneurs plus en détail.

Étape 5 – Exécution d'un conteneur Docker

Le conteneur `hello-world` que vous avez effectué à l'étape précédente est un exemple de conteneur qui fonctionne et qui quitte après avoir émis un message de test. Les conteneurs peuvent être beaucoup plus utiles que cela, et ils peuvent être interactifs. Après tout, ils sont similaires aux machines virtuelles, mais ils sont plus économes en ressources.

À titre d'exemple, exécutons un conteneur en utilisant la dernière image d'Ubuntu. La combinaison des commutateurs **-i** et **-t** vous donne un accès interactif au shell dans le conteneur :

```
$ docker run -it ubuntu
```
Copie

Votre invitation de commande devrait changer pour permettre le fait que vous travaillez maintenant à l'intérieur du conteneur et devrait prendre cette forme :

```
Output
root@d9b100f2f636:/#
```

Notez l'identifiant du conteneur dans l'invitation de commande. Dans cet exemple, il s'agit de `d9b100f2f636`. Vous aurez besoin de cet ID de conteneur plus tard pour identifier le conteneur lorsque vous voudrez le supprimer.

Vous pouvez maintenant n'importer quelle commande à l'intérieur du conteneur. Mettons par exemple à jour la base de données des paquets à l'intérieur du conteneur. Vous ne devez pas préfixer une commande avec `sudo`, car vous opérez à l'intérieur du conteneur en tant qu'utilisateur **root** :

```
root@d9b100f2f636:/# apt update
```
Copie

Ensuite, installez n'importe quelle application dans le conteneur. Installons Node.js :

```
root@d9b100f2f636:/# apt install nodejs
```
Copie

Ceci installe Node.js dans le conteneur à partir du dépôt officiel d'Ubuntu. Une fois l'installation terminée, vérifiez que Node.js est installé :

```
root@d9b100f2f636:/# node -v
```
Copie

Vous voyez le numéro de version affiché dans votre terminal :

```
Output
v10.19.0
```

Les modifications que vous apportez à l'intérieur du conteneur ne s'appliquent qu'à ce conteneur.

Pour quitter le conteneur, marqué `exit` à l'invitation.

Voyons maintenant comment gérer les conteneurs sur notre système.

Étape 6 – Gestion des conteneurs Docker

Après avoir utilisé Docker pendant un certain temps, vous aurez de nombreux conteneurs actifs (en cours d'exécution) et inactifs sur votre ordinateur. Pour voir les **actifs**, utilisez :

```
$ docker ps
```
Copie

Vous verrez une sortie similaire à celle-ci :

```
Output
CONTAINER ID        IMAGE               COMMAND             CREATED
```

Dans ce tutoriel, vous avez lancé deux conteneurs ; un à partir de l'image `hello-world` et un autre à partir de l'image `ubuntu`. Les deux conteneurs ne sont plus actifs, mais ils existent toujours sur votre système.

Pour voir tous les conteneurs, actifs et inactifs, actualisés `docker ps` avec le commutateur `-a` :

```
$ docker ps -a
```
Copie

Vous voyez une sortie semblable à celle-ci :

```
1c08a7a0d0e4        ubuntu              "/bin/bash"         2 minutes ago       Exited (0) 8 seconds a
a707221a5f6c        hello-world         "/hello"            6 minutes ago       Exited (0) 6 minutes a
```

Pour voir le dernier conteneur que vous avez créé, passez-le au commutateur `-l` :

```
$ docker ps -l
```

Copie

```
$ CONTAINER ID        IMAGE               COMMAND             CREATED             STATUS
$ 1c08a7a0d0e4        ubuntu              "/bin/bash"         2 minutes ago       Exited (0) 40 secon
$
```

Copie

Pour télécharger un conteneur arrêté, utiliser `docker start`, suivre l'ID du conteneur ou de son nom. Démarrons le conteneur basé sur Ubuntu avec l'ID de `1c08a7a0d0e4` :

```
$ docker start 1c08a7a0d0e4
```

Copie

Le conteneura, et vous pouvez utiliser `docker ps` pour voir son statut

```
Output
CONTAINER ID        IMAGE               COMMAND             CREATED             STATUS              PO
1c08a7a0d0e4        ubuntu              "/bin/bash"         3 minutes ago       Up 5 seconds
```

Pour utiliser un conteneur en cours d'exécution, `docker stop` suivi de l'ID ou du nom du conteneur. Cette fois, nous utiliserons le nom que Docker a attribué au conteneur, qui est `quizzical_mcnulty` :

```
$ docker stop quizzical_mcnulty
```

Copie

Une fois que vous avez décidé que vous n'avez plus besoin d'un conteneur, différencié-le avec la commande `docker rm`, en utilisant à nouveau l'ID ou le nom du conteneur. Utilisez la commande `docker ps -a` pour trouver l'ID ou le nom du conteneur associé à l'image `hello-world` et supprimez-le.

```
$ docker rm youthful_curie
```

Copie

Vous pouvez démarrer un nouveau conteneur et lui donner un nom en utilisant le commutateur `--name`. Vous pouvez également utiliser le sélecteur `--rm` pour créer un conteneur qui se supprime de lui-même lorsqu'il est arrêté. Voir la commande `docker run help` pour plus d'informations sur ces options et d'autres.

Les conteneurs peuvent être transformés en images que vous pouvez utiliser pour construire de nouveaux conteneurs. Voyons comment cela fonctionne.

Étape 7 – Transformation d'un conteneur en une image Docker

Lorsque vous démarrez une image Docker, vous pouvez créer, modifier et supprimer des fichiers comme vous le pouvez avec une machine virtuelle. Les modifications que vous apportez ne s'appliqueront qu'à ce conteneur. Vous pouvez le démarrer et l'arrêter, mais une fois que vous l'aurez détruit avec la commande `docker rm`, les modifications seront perdues pour de bon.

Cette section vous montre comment enregistrer l'état d'un conteneur en tant que nouvelle image Docker.

Après avoir installé Node.js dans le conteneur Ubuntu, vous avez maintenant un conteneur qui s'exécute à partir d'une image, mais le conteneur est différent de l'image que vous avez utilisée pour le créer. Mais vous pourriez vouloir réutiliser ce conteneur Node.js comme base pour de nouvelles images plus tard.

Ensuite, effectuez les modifications dans une nouvelle instance d'image Docker à l'aide de la commande suivante.

```
$ docker commit -m "What you did to the image" -a "Author Name" container_id repository/new_
```

Copy

LIÉ

Comment installer un discours sur Ubuntu 18.04

Didacticiel

Comment installer la pile Linux, Apache, MySQL, PHP (LAMP) sur Ubuntu 20.04 [Démarrage rapide]

Didacticiel

Le commutateur **-m** est destiné au message de validation qui vous aide, ainsi que les autres, à connaître les modifications que vous avez apportées, tandis que **-a** est utilisé pour spécifier l'auteur. Le `container_id` est celui que vous avez noté plus tôt dans le tutoriel lorsque vous avez lancé la session interactive de Docker. À moins de créer des référentiels supplémentaires sur Docker Hub, le `référentiel` est généralement votre nom d'utilisateur Docker Hub.

Par exemple, pour l'utilisateur **sammy**, avec l'ID de conteneur `d9b100f2f636`, la commande serait :

```
$ docker commit -m "added Node.js" -a "sammy" d9b100f2f636 sammy/ubuntu-nodejs
```

Copy

Lorsque vous *validez* une image, la nouvelle image est enregistrée localement sur votre ordinateur. Plus loin dans ce tutoriel, vous apprendrez comment pousser une image vers un registre Docker comme Docker Hub pour que d'autres puissent y accéder.

L'énumération des images Docker affichera à nouveau la nouvelle image, ainsi que l'ancienne image dont elle est issue :

```
$ docker images
```

Copy

Vous verrez une sortie de ce type :

```
Output
REPOSITORY              TAG                 IMAGE ID            CREATED             SIZE
sammy/ubuntu-nodejs     latest              7c1f35226ca6        7 seconds ago       179MB
...
```

Dans cet exemple, `ubuntu-nodejs` est la nouvelle image, qui a été dérivée de l'image `ubuntu` existante à partit de Docker Hub. La différence de taille reflète les modifications apportées. Et dans cet exemple, le changement est que NodeJS a été installé. Donc la prochaine fois que vous aurez besoin d'exécuter un conteneur en utilisant Ubuntu avec NodeJS pré-installé, vous pourrez simplement utiliser la nouvelle image.

Vous pouvez également construire des images à partir d'un `Dockerfile`, qui vous permet d'automatiser l'installation de logiciels dans une nouvelle image. Cependant, cela n'entre pas dans le cadre de ce tutoriel.

Partageons maintenant la nouvelle image avec d'autres personnes afin qu'elles puissent créer des conteneurs à partir de celle-ci.

Étape 8 – Pousser des images Docker dans un référentiel Docker

L'étape logique suivante après la création d'une nouvelle image à partir d'une image existante est de la partager avec quelques amis choisis, le monde entier sur Docker Hub, ou tout autre registre Docker auquel vous avez accès. Pour pousser une image vers Docker Hub ou tout autre registre Docker, vous devez y avoir un compte.

Cette section vous montre comment pousser une image Docker vers Docker Hub. Pour apprendre comment créer votre propre registre Docker privé, consultez Comment configurer un registre Docker privé sur Ubuntu 14.04.

Pour pousser votre image, connectez-vous d'abord à Docker Hub.

```
$ docker login -u docker-registry-username
```

Copy

Vous serez invité à vous s'authentifier à l'aide de votre mot de passe Docker Hub. Si vous avez spécifié le bon mot de passe, l'authentification devrait réussir.

**Remarque **: Si votre nom d'utilisateur du registre Docker est différent du nom d'utilisateur local que vous avez utilisé pour créer l'image, vous devrez tagger votre image avec votre nom d'utilisateur du registre. Pour l'exemple donné à la dernière étape, vous devriez taper :

```
$ docker tag sammy/ubuntu-nodejs docker-registry-username/ubuntu-nodejs
```

Copy

Ensuite, vous pouvez pousser votre propre image à l'aide de :

```
$ docker push docker-registry-username/docker-image-name
```
Copy

Pour pousser l'image `ubuntu-nodejs` vers le référentiel **sammy**, la commande serait :

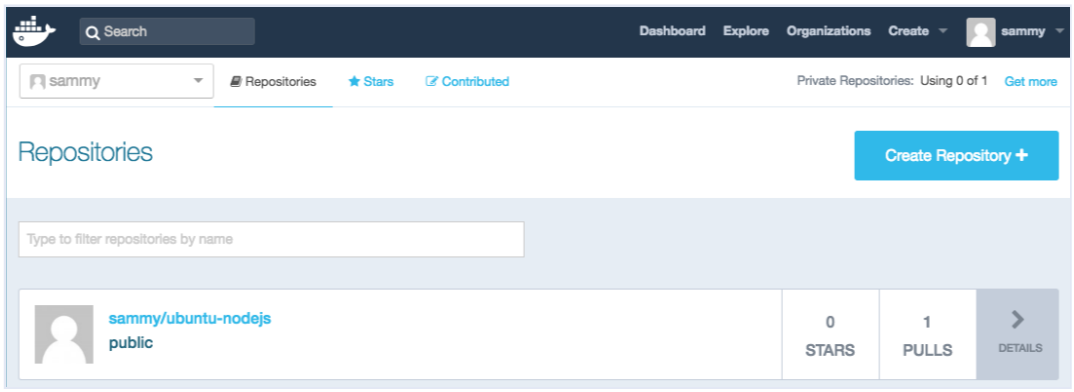
```
$ docker push sammy/ubuntu-nodejs
```
Copy

Le processus peut prendre un certain temps pour s'achever car il télécharge les images, mais une fois terminé, la sortie ressemblera à ceci :

```
Output
The push refers to a repository [docker.io/sammy/ubuntu-nodejs]
e3fbbfb44187: Pushed
5f70bf18a086: Pushed
a3b5c80a4eba: Pushed
7f18b442972b: Pushed
3ce512daaf78: Pushed
7aae4540b42d: Pushed

...
```

Après avoir poussé une image vers un registre, elle doit être répertoriée sur le tableau de bord de votre compte, comme le montre l'image ci-dessous.



Si une tentative de push entraîne une erreur de ce type, c'est que vous ne vous êtes probablement pas connecté :

```
Output
The push refers to a repository [docker.io/sammy/ubuntu-nodejs]
e3fbbfb44187: Preparing
5f70bf18a086: Preparing
a3b5c80a4eba: Preparing
7f18b442972b: Preparing
3ce512daaf78: Preparing
7aae4540b42d: Waiting
unauthorized: authentication required
```

Connectez-vous avec le `docker login` et répétez la tentative de poussée. Vérifiez ensuite qu'elle existe sur votre page de dépôt Docker Hub.

Vous pouvez maintenant utiliser `docker pull sammy/ubuntu-nodejs` pour tirer l'image vers une nouvelle machine et l'utiliser pour lancer un nouveau conteneur.

Conclusion

Dans ce tutoriel, vous avez installé Docker, travaillé avec des images et des conteneurs, et poussé une image modifiée sur Docker Hub. Maintenant que vous connaissez les bases, explorez les autres tutoriels de Docker dans la communauté DigitalOcean.

À propos des auteurs

Brian Hogan Auteur

Développeur et auteur chez DigitalOcean.

commentaires

1 commentaires

B I U H1 H2 H3 1.

Laissez un commentaire...

Connectez-vous pour commenter

phénixibm • 24 décembre 2020

Magnifique!!!

Réponse