

Administration de Docker

Document reprenant les étapes de configurations et administrations concernant Docker



Référence : EF-VIRTU-DOCKER

Auteur(s) :

Dorian Manzanares
Alex Falzon

Destinataire(s) :

Easyformer

Date de modification : 08/11/21

Version : 1

Sommaire

page

1	INTRODUCTION	4
1.1	HISTORIQUE DE LA VIRTUALISATION	4
1.2	TIME SHARING	4
1.3	MULTI PROGRAMMING	4
2	LA VIRTUALISATION	6
2.1	LES TYPES DE VIRTUALISATIONS	6
2.1.1	Serveur	6
2.1.2	Application	6
2.1.3	Postes de travail	6
2.1.4	Stockage	6
2.2	LES TECHNOLOGIES DE VIRTUALISATION	7
2.2.1	Complète	7
2.2.2	Hybride	7
2.2.3	Applicative	7
2.2.4	Para-virtualisation	8
2.2.5	HAL	8
2.2.6	Virtualisation assistée par matériel	9
3	DOCKER	10
3.1	INTRODUCTION	10
3.1.1	Présentation de Docker	10
3.1.2	Principe de Docker	10
3.2	INSTALLATIONS ET CONFIGURATIONS	11
3.2.1	Prérequis	11
3.2.2	Bonnes pratiques	11
3.2.3	Ubuntu	11
3.2.4	Installation sous Linux	12
3.2.5	Docker	13
3.2.6	Docker-hub	15
3.3	UTILISATION DE DOCKER	16
3.3.1	Les commandes informatives	16
3.3.2	Les commandes gestion d'images	17
3.3.3	Les commande de gestion des conteneurs	17
	Run	17
	Exec	18
	Log	18
	Commit	19
	Autres commandes de gestion des conteneurs	20
3.4	INTERFACE GRAPHIQUE	21
3.4.1	Shipyard	21
	Proxy	22
	Contrôleur	23
3.4.2	Portainer	25
3.5	GESTION DES CONTENEURS	26
3.5.1	Images Docker	26
3.5.2	Conteneurs	29
	29
3.5.3	Gérer les conteneurs et images	30
	Validation des modifications	32
3.5.4	Exporter une image dans le Docker hub	33
3.5.5	Communication entre conteneurs	34
3.5.6	Monter un répertoire hôte dans un conteneur	34



3.5.7	Conteneur de données.....	35
3.5.8	Swarm	36
	Swarm manager.....	37
	Swarm Agent	38
3.5.9	Kubernetes	39
3.6	INSTALLATION ET CONFIGURATION DE KUBERNETES	39
3.6.1	Introduction.....	39
3.6.2	Préparation des hôtes	40
	Désactiver la Swap	41
3.6.3	Installation des Kubeadm.....	41
3.6.4	Initialisation du cluster.....	42
	Ajout des nœuds de travail au cluster	46
	Réalisation des tests	47
4	OUTILS DOCKER	54
4.1	ORCHESTRATION ET ORDONNANCEUR.....	54
4.2	INTEGRATION CONTINUE	54
4.3	SURVEILLANCE	54
4.4	ENREGISTREMENT.....	54
4.5	SECURITE	54
4.6	STOCKAGE.....	54
4.7	MISE EN RESEAU.....	54
4.8	DECOUVERTE DE SERVICE	54
4.9	BUILD.....	54
4.10	GESTION GRAPHIQUE	54



1 Introduction

1.1 Historique de la virtualisation

Bien que la virtualisation soit très à la mode de nos jours et au cœur des préoccupations de la majeure partie des services informatiques, ce n'est pas un concept nouveau.

A l'origine les premiers ordinateurs conçus ne permettaient l'exécution que d'une seule application à la fois.

Dans les années 1960, IBM développe la première technologie de virtualisation permettant à une plateforme mainframe (super-ordinateur) d'exécuter au sein de pseudo-machines (machines virtuelles aujourd'hui) plusieurs instances d'application en parallèle, sous le contrôle d'un programme (hyperviseur aujourd'hui) dont la fonction était d'allouer les ressources et isoler ces instances.

Les notions suivantes voient le jour...

1.2 Time sharing

Le temps partagé ou pseudo-parallélisme¹, est une approche permettant de simuler le partage par plusieurs utilisateurs de temps processeur. Il ne faut pas le confondre avec le terme de multitâche : un système peut être multitâche sans être à temps partagé (par exemple s'il dispose de pilotes effectuant des tâches de fond asynchrones) ; il a également existé quelques systèmes de temps partagé qui n'étaient pas multitâches : le processeur divisait simplement son temps en tranches fixes.



1.3 Multi programming

En 1978, l'architecture X86 qui désigne la famille de microprocesseurs utilisant les jeux d'instructions du même nom fait son apparition et s'impose durant la décennie suivante comme le modèle à suivre. De nos jours encore le X86 équipe la majorité des microprocesseurs à destination des postes de travail et des serveurs.

X86 est un terme que l'on retrouve souvent dans les documents de virtualisation, ici il désigne une famille de processeurs qui équipent un grand nombre de nos équipements informatiques (postes de travail, serveurs, ...). Il s'est imposé comme un modèle à suivre et on retrouve cette architecture de puces aujourd'hui dans la majorité des processeurs vendu par les fabricants (Intel, AMD, Via).

Ce jeu d'instructions s'appuie à l'origine sur le CISC (Complex Instruction Set Computer) bien qu'il tende à évoluer vers le RISC (Reduced Instruction Set Computer) sur les dernières puces. On trouve du RISC dans la plupart des processeurs ARM de nos équipements informatiques mobiles (téléphones, Tablettes, ...).

En 1999 VMWARE invente une solution de virtualisation pour les architectures X86 pour répondre principalement aux contraintes de sous-utilisation des ressources matérielles et la popularise via la sortie de son premier produit Vmware Workstation 1.0.

En 2005, Intel-VT puis en 2006, AMD-V) sont les 2 technologies de virtualisation matérielle assistée, développées par Intel et AMD pour répondre aux contraintes de privilèges des architectures X86.



2 La virtualisation

2.1 Les types de virtualisations

La virtualisation est l'ensemble des techniques matérielles et logicielles permettant de fournir un ensemble ou sous-ensemble de ressources informatiques de manière qu'elles puissent être utilisées, avec avantages, de manière indépendante de la plateforme matérielle.

Il existe différents types de virtualisation :

- La virtualisation de serveur
- La virtualisation d'application
- La virtualisation de postes de travail
- La virtualisation de stockage

2.1.1 Serveur

La virtualisation de serveur permet de regrouper plusieurs serveurs physiques sous-employés sur un seul hôte qui exécute des systèmes virtuels. Il permet aussi de réduire la consommation électrique et le nombre d'administrateurs. Il participe beaucoup à la réalisation des économies (locaux, consommation électrique).

2.1.2 Application

Elle permet de séparer complètement l'application du système d'exploitation hôte et des autres applications présentes afin d'éviter les conflits. En outre elle peut être définie comme la technologie qui permet de séparer l'environnement du bureau et des applications associées de la machine physique.

2.1.3 Postes de travail

La virtualisation des postes de travail facilite le travail des administrateurs systèmes et réseaux. Un poste de travail virtualisé ou bureau virtuel peut être hébergé soit directement sur l'ordinateur du client soit sur un serveur dans le centre de données.

2.1.4 Stockage

La virtualisation des stockages permet :

- D'exploiter au maximum les ressources,
- D'exploiter au mieux le stockage des disques durs,
- Centraliser et sécuriser les données

Le centre de données s'articule autour d'un SAN (Storage Area Network).

Un des plus grands défis de la virtualisation reste le stockage.

Dans les faits, c'est le plus souvent le stockage qui fait exploser les coûts, et crée des engorgements et des pertes de performances.



2.2 Les technologies de virtualisation

2.2.1 Complète

La virtualisation complète est une virtualisation dans laquelle le système d'exploitation invité ne sait pas qu'il se trouve dans un environnement virtualisé et, par conséquent, le matériel est virtualisé par le système d'exploitation hôte, de sorte que l'invité peut émettre des commandes sur ce qu'il considère être du matériel réel, mais qu'il ne fait que simuler les périphériques matériels créés par l'hôte.

2.2.2 Hybride

Il existe également une combinaison de virtualisation complète et de paravirtualisation appelée virtualisation hybride, dans laquelle des parties du système d'exploitation invité utilisent la paravirtualisation de certains pilotes matériels et l'hôte utilise la virtualisation complète pour d'autres fonctionnalités. Cela produit souvent des performances supérieures sur l'invité sans qu'il soit nécessaire de complètement paravirtualiser l'invité.

Par exemple l'invité utilise la virtualisation complète pour les instructions privilégiées dans le noyau, mais la paravirtualisation pour les demandes d'E/S (Entrées/Sorties) utilisant un pilote spécial dans l'invité. De cette façon, le système d'exploitation invité n'a pas besoin d'être complètement paravirtualisée, car cela n'est parfois pas disponible, mais peut toujours profiter de certaines fonctionnalités paravirtualisées en implémentant des pilotes spéciaux pour l'invité.

2.2.3 Applicative

Le principe de la virtualisation applicative consiste à générer un exécutable à partir du programme d'installation (par exemple Word). Celui-ci se lance alors physiquement en local (et non pas depuis un autre point du réseau) sans que l'on installe quoi que soit.

Softgrid, Altiris, Thinstall, Citrix proposent différentes variantes de ce principe. La totalité des programmes et fichiers nécessaires se trouvent sur un espace de stockage quelconque (clé USB, mais aussi CD, email, unité disque locale ou réseau).

Le logiciel de virtualisation se positionne entre le système et l'application et fournit à cette dernière la totalité des ressources d'environnement qui lui sont nécessaires pour fonctionner, sans qu'elle n'accède jamais au système réel (donc sans impacter la stabilité du poste hôte et sans altérer ses registres systèmes).

Par exemple OpenOffice virtualisé avec ThinStall n'occupe que la moitié d'une clé USB 256 Mo et fonctionne immédiatement sur n'importe quelle machine. L'application, bien qu'un peu ralentie, reste parfaitement utilisable.

L'installation d'une nouvelle application sur un poste est considérablement raccourcie et simplifiée. De plus, rien n'est modifié sur la machine et il ne restera aucune trace après le travail.

Le système permet notamment d'utiliser les applications de l'entreprise à l'extérieur, en bénéficiant des liaisons performantes locales sans menacer la sécurité de la machine utilisée ni les données de l'entreprise.



2.2.4 Para-virtualisation

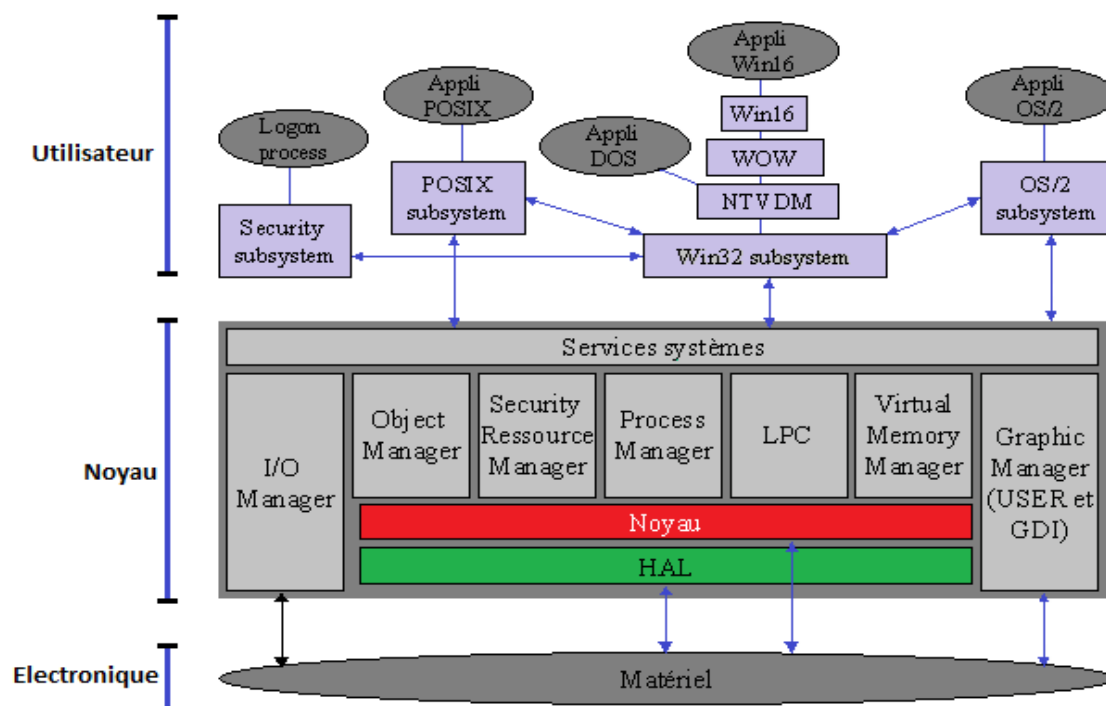
La paravirtualisation est une virtualisation dans laquelle le système d'exploitation invité (celui qui est virtualisé) sait qu'il s'agit d'un invité et possède donc des pilotes qui, au lieu d'émettre des commandes matérielles, émettent simplement des commandes directement au système d'exploitation hôte. Cela inclut également la gestion de la mémoire et des threads, qui nécessitent généralement des instructions privilégiées non disponibles dans le processeur.

2.2.5 HAL

La HAL pour Hardware Abstraction Layer, ou en français, Couche d'Abstraction Matérielle. C'est grâce à cette couche que Windows NT est portable sur plusieurs types de machines. Son rôle est de masquer complètement la partie matérielle au système d'exploitation lui-même. Tout matériel va être représenté virtuellement au système d'exploitation. Le système d'exploitation va utiliser le matériel via une interface unique, sans se soucier du matériel. C'est à la couche d'abstraction matérielle de traduire les demandes au système physique. A un type de machine correspond une couche HAL spécifique. Grâce à cette couche, Windows NT peut tourner sur plusieurs types de microprocesseurs : Intel, MIPS, PowerPC, Alpha.

C'est encore la couche HAL qui gère les systèmes multi-processeurs, en intégrant l'interface SMP (Symmetric Multi Processing). La couche HAL est différente suivant le nombre de processeurs dans le système.

La couche HAL représente chaque processeur comme un processeur virtuel au noyau. Le noyau, on le verra plus tard, dispatchera ses tâches sur ces différents processeurs. Que ce soient des processeurs Alpha, Intel ou autre, le noyau voit toujours des processeurs virtuels. La couche HAL n'est accessible que par le NT Executive. Pour accéder au matériel, les programmes utilisateurs sont donc obligés de passer par les Native APIs (enfin presque).



2.2.6 Virtualisation assistée par matériel

La virtualisation assistée par le matériel est un type de virtualisation complète dans laquelle l'architecture du microprocesseur comporte des instructions spéciales pour faciliter la virtualisation du matériel. Ces instructions peuvent permettre de configurer un contexte virtuel afin que l'invité puisse exécuter des instructions privilégiées directement sur le processeur sans affecter l'hôte. Un tel ensemble de fonctionnalités est souvent appelé un hyperviseur. Si ces instructions n'existent pas, la virtualisation complète est toujours possible. Toutefois, elle doit être effectuée à l'aide de techniques logicielles telles que la recompilation dynamique (translation binaire), où l'hôte recompile à la volée des instructions privilégiées de l'invité pour pouvoir s'exécuter de manière non privilégiée sur le serveur hôte. Dans le cas de la virtualisation assistée par matériel, la virtualisation est conçue dans le jeu d'instructions fournit des instructions pour partitionner l'hôte. Voir la technologie Intel VT-x ou AMD-V à titre d'exemple. Pour que l'hyperviseur fonctionne directement avec le matériel sans utiliser de système d'exploitation, il fournit une virtualisation complète.



3 Docker

3.1 Introduction

3.1.1 Présentation de Docker

Le développement de Docker a été lancé par Solomon Hykes en tant que projet interne chez dotCloud, une entreprise en tant que PaaS (plate-forme en tant que service), mais le logiciel est maintenant géré par la communauté des dockers et Docker Inc.

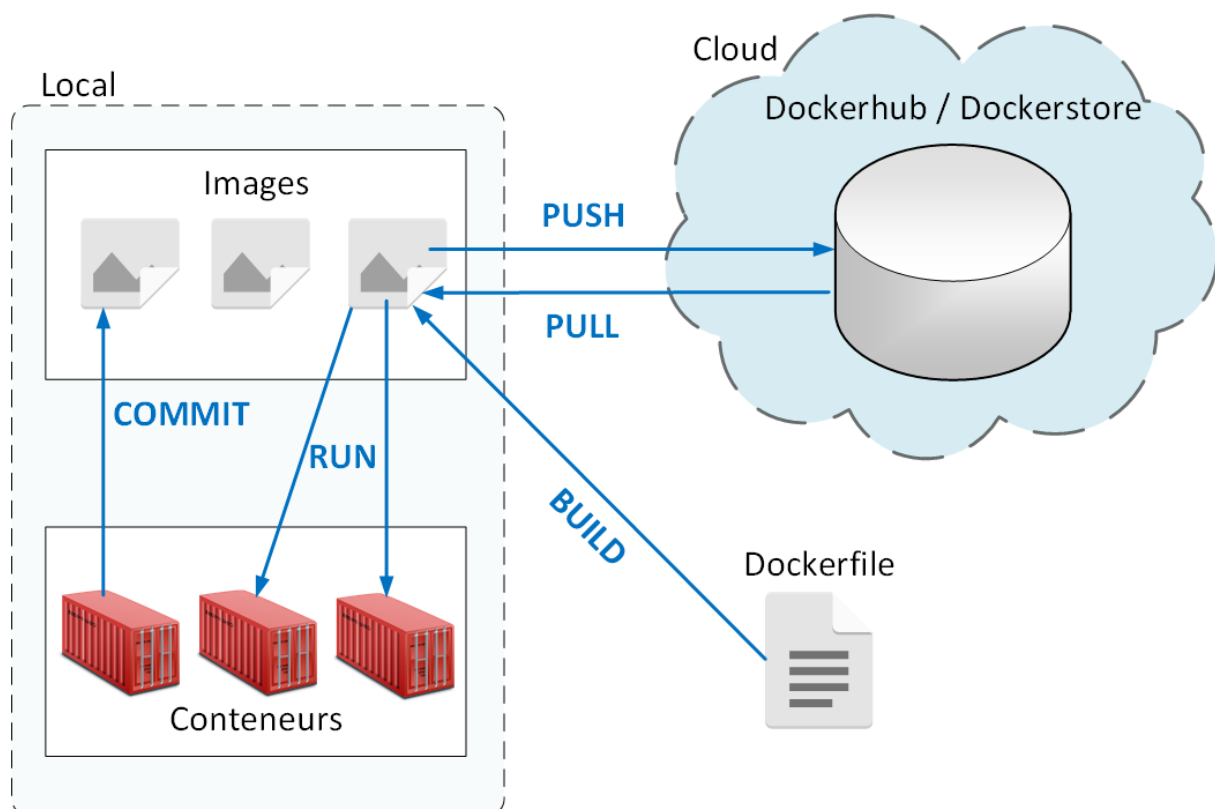
Docker est un projet open source qui fournit une plate-forme ouverte aux développeurs et aux administrateurs système pour :

- La création,
- La mise en package
- Et l'exécution d'applications en tant que conteneur léger.

Docker automatise le déploiement d'applications à l'intérieur de conteneurs de logiciels.

3.1.2 Principe de Docker

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 conviviaux et plus dépendants du système d'exploitation hôte.



3.2 Installations et configurations

3.2.1 Prérequis

Docker nécessite une architecture 64 bits pour l'installation et le noyau Linux doit être 3.10 ou plus récent. Je vais utiliser Ubuntu 18.04 LTS (Bionic Beaver) ici avec la version 4.15.0 du noyau.

3.2.2 Bonnes pratiques

Il est préconisé :

- De prévoir un volume dédié pour le stockage de Docker.
- D'Interdire les communications entre les containers.

Pour de dernier point modifier la variable DOCKER_OPTS dans le fichier /etc/default/docker comme ceci :

```
DOCKER_OPTS="--icc=false"
```

3.2.3 Ubuntu

Dans le cadre de cet ouvrage nous allons réaliser une installation sous Ubuntu.

Voici quelques rappels :

Taper pour connaître le nom des interface réseau :

```
lshw -C network
```

Puis gérer l'adressage avec le « netplan » :

```
nano /etc/netplan/*.yaml
```

Voici un exemple de configuration réseau :

```
network:
version: 2
ethernets:
ens33:
dhcp4: no
# dhcp6: no
addresses: [192.168.8.253/24]
gateway4: 192.168.8.254
nameservers:
addresses: [8.8.8.8,8.8.4.4]
ens34:
dhcp4: no
addresses: [10.10.10.254/26]
```



Appliquez avec :

```
netplan apply
```

Passons sur l'activation de l'utilisateur « root » avec « sudo su » puis « passwd root » et sur l'activation de openssh et concentrons-nous maintenant sur Docker.

3.2.4 Installation sous Linux

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

```
sudo apt update
```

Ensuite, installez quelques packages prérequis qui permettent d'utiliser des packages via HTTPS:

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

Ajoutez ensuite la clé GPG du référentiel officiel de Docker à votre système :

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

Ajoutez le référentiel Docker aux sources APT:

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

Ensuite, mettez à jour la base de données de packages avec les packages Docker du référentiel nouvellement ajouté :

```
sudo apt update
```

Assurez-vous que vous êtes sur le point d'installer à partir du dépôt Docker au lieu du dépôt Ubuntu par défaut :

```
apt-cache policy docker-ce
```

Vous verrez une sortie comme celle-ci, bien que le numéro de version de Docker puisse être différent :

```
docker-ce:  
Installed: (none)  
Candidate: 18.03.1~ce~3-0~ubuntu  
Version table:  
18.03.1~ce~3-0~ubuntu 500  
500 https://download.docker.com/linux/ubuntu bionic/stable amd64 Packages
```

Enfin, installez Docker :

```
sudo curl -sSL get.docker.com | sh
```

Docker devrait maintenant être installé, le démon démarré et le processus activé pour démarrer au démarrage. Vérifiez qu'il fonctionne :

```
sudo systemctl status docker
```



La sortie doit être similaire à la suivante, indiquant que le service est actif et en cours d'exécution :

```
docker.service - Docker Application Container Engine
Loaded: loaded (/lib/systemd/system/docker.service; enabled; vendor preset:
enabled)
Active: active (running) since Thu 2018-07-05 15:08:39 UTC; 2min 55s ago
Docs: https://docs.docker.com
Main PID: 10096 (dockerd)
Tasks: 16
CGroup: /system.slice/docker.service
└─10096 /usr/bin/dockerd -H fd://
└─10113 docker-containerd --config /var/run/docker/containerd/containerd.toml
```

3.2.5 Docker

La première étape consistera à mettre à jour notre système grâce aux commandes suivante

```
sudo apt update && apt upgrade
```

Ensuite nous allons installer docker, rendez-vous à la racine puis :

```
sudo curl -sSL get.docker.com | sh
```

Maintenant nous allons faire en sorte de démarrer automatiquement Docker à chaque nouveau reboot ou démarrage du système

```
sudo systemctl enable docker
```

Enfin nous démarrons notre service

```
sudo systemctl start docker
```

Et nous vérifions que tout est bien démarré :

```
sudo systemctl status docker
```

La sortie doit être similaire à la suivante, indiquant que le service est actif et en cours d'exécution :

```
docker.service - Docker Application Container Engine
Loaded: loaded (/lib/systemd/system/docker.service; enabled; vendor preset:
enabled)
Active: active (running) since Thu 2018-07-05 15:08:39 UTC; 2min 55s ago
Docs: https://docs.docker.com
Main PID: 10096 (dockerd)
Tasks: 16
CGroup: /system.slice/docker.service
└─10096 /usr/bin/dockerd -H fd://
└─10113 docker-containerd --config /var/run/docker/containerd/containerd.toml
```



Par défaut, la commande docker ne peut être exécutée que par l'utilisateur root ou par un utilisateur du groupe docker, qui est automatiquement créé lors du processus d'installation de Docker. Si vous essayez d'exécuter la commande docker sans la préfixer avec « sudo » ou sans être dans le groupe docker, vous obtiendrez une sortie comme celle-ci :

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

Si vous souhaitez éviter de taper « sudo » à chaque fois que vous exécutez la commande « docker », ajoutez votre nom d'utilisateur au groupe docker :

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

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

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

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

Confirmez que votre utilisateur est maintenant ajouté au groupe Docker en tapant :

```
id -nG
```

Ce qui retourne :

```
${USER} sudo docker
```

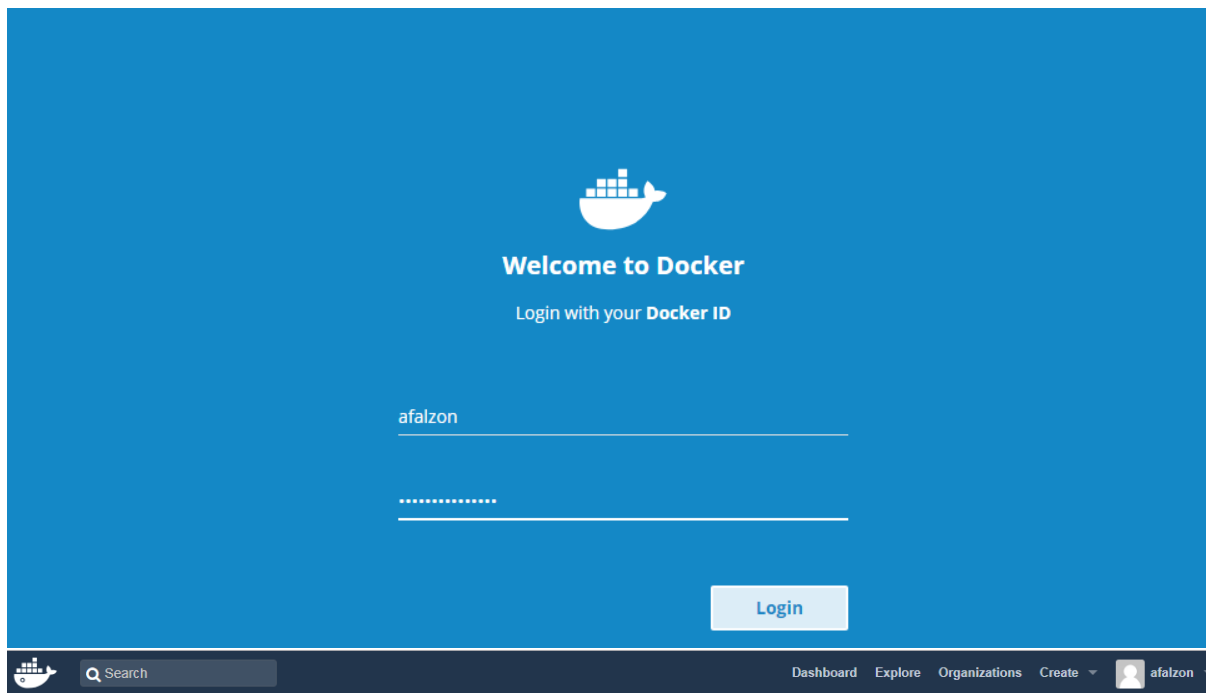
Si vous devez ajouter un utilisateur au groupe docker avec lequel vous n'êtes pas connecté, déclarez-le explicitement en utilisant :

```
sudo usermod -aG docker username
```

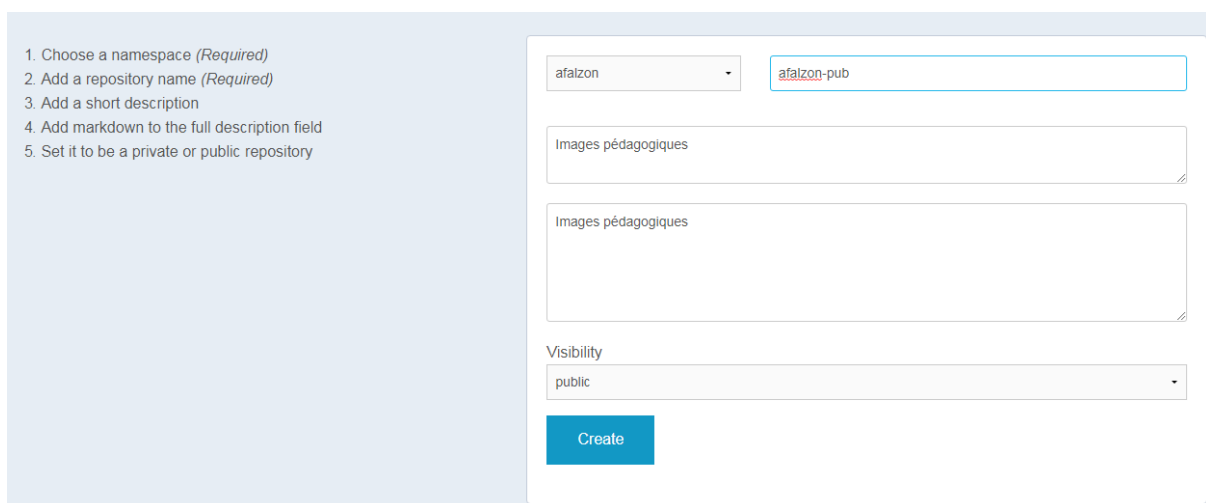


3.2.6 Docker-hub

Afin de pouvoir pousser vos images dans le Dockerhub vous devez au préalable créer un compte sur celui-ci. Rendez-vous au <https://hub.docker.com/>



Create Repository



Plus tard dans cet ouvrage nous utiliserons la commande suivante pour connecter Docker au dépôt :

```
Docker login -u <login-docker-hub> -p <password-docker-hub>
```



3.3 Utilisation de Docker

Pour voir la liste des commandes tapez simplement :

```
$ docker
```

Pour voir l'aide d'une sous-commande tapez :

```
$ docker <sous-commande> --help
```

Voici la liste des sous-commandes francisées et regroupées en 3 catégories :

- Informatives,
- Gestion d'images,
- Gestion de conteneurs

Celle-ci seront détaillées plus loin dans la mise en pratique de cet ouvrage

3.3.1 Les commandes informatives

event Obtenir des événements en temps réel à partir du serveur

info Afficher des informations à l'échelle du système

inspect Retourne les informations de bas niveau sur les objets Docker

logs Récupère les journaux d'un conteneur

stats Afficher un flux en direct de statistiques sur l'utilisation des ressources des conteneurs

port Liste des mappages ou un mappage spécifique pour le conteneur

version Afficher les informations de version de Docker

top Affiche les processus en cours d'exécution d'un conteneur

diff Inspecter les modifications apportées aux fichiers ou aux répertoires sur le système de fichiers d'un conteneur

login ajoute ses identifiants du DHR (Docker Hub Registry) via ligne de commandes

search Recherche un conteneur sur le Docker Hub



3.3.2 Les commandes gestion d'images

Images Liste d'images

history Afficher l'histoire d'une image

pull Tirez une image ou un référentiel d'un registre

commit Créer une nouvelle image à partir des modifications d'un conteneur

build Construire une image à partir d'un fichier Docker

tag Crée une balise TARGET_IMAGE faisant référence à SOURCE_IMAGE

rmi Supprimer une ou plusieurs images

export Exporter le système de fichiers d'un conteneur en tant qu'archive tar

import Importe le contenu d'une archive pour créer une image de système de fichiers

save Sauvegarde une ou plusieurs images dans une archive tar (diffusée par défaut sur STDOUT)

load Charge une image depuis une archive tar ou STDIN

login Connectez-vous à un registre Docker

logout Déconnexion d'un registre Docker

push Poussez une image ou un référentiel dans un registre

search dans le Docker Hub pour les images

3.3.3 Les commande de gestion des conteneurs

Run

La commande « run » permet d'instancier un nouveau container à partir d'une image.

Exécuter directement une instruction dans un nouveau container :

```
docker run ubuntu:14.04 echo "hello world"
```

Affiche :

```
hello world
```

Utiliser un port aléatoire pour la redirection :

```
docker run -P zopnow/lamp-stack
```

Configurer la redirection de port entre le container et la machine hôte :

```
docker run -p 8080:80 zopnow/lamp-stack
```

Note : ici le port 80 correspond au port du container, le port 8080 correspond au port de la machine hôte.

Certaines applications dans un container tournent en tâche de fond, or le container fini de s'exécuter quand il n'y a plus d'application au premier plan. Pour résoudre ce problème, il suffit d'exécuter le container en tant que service (daemon).



Pour exécuter un container en tant que service, il faut utiliser l'option -d :

```
docker run -d ubuntu:14.04 /bin/sh -c "while true; do echo hello world; sleep 1; done"
```

Ici le conteneur affiche toutes les secondes "hello world" dans la sortie standard du container

Pour installer une application dans un conteneur il faut :

- soit exécuter l'installation directement (rapide mais limité),
- soit exécuter l'installation via un pseudo tty interactif.

Installer une application directement :

Note : -y permet d'exécuter l'installation en mode « noninteractive ». Quand l'installation est complexe, il faut utiliser la console interactive avec l'option « -t »

```
docker run ubuntu:14.04 apt-get install -y ping
```

Installer une application via un pseudo tty interactif

```
docker run -t -i ubuntu:14.04 /bin/bash
```

Puis dans l'invite de commande :

```
sudo apt-get install ping
```

Exec

La commande « exec » permet d'exécuter une commande dans un container en cours d'exécution :

```
docker exec 32a6485f apt-get install ping
```

Log

Afficher la sortie standard d'un « daemon » en cours d'exécution :

```
docker logs sharp_lalande
```

Note : ici "sharp_lalande" est le nom du container daemon, l'option -f permet d'afficher les retours de la ligne de commande en temps réel (à la manière d'un « tail -f »).



Commit

Prenons une image ubuntu de base à laquelle on ajoute le paquet "ping" :

```
docker run ubuntu:14.04 apt-get install ping
```

Si on souhaite sauvegarder cette image d'ubuntu avec le paquet "ping", il faut récupérer l'ID du nouvel container :

```
docker ps -l
```

Affiche l'ID "6982a9948422..."

et commit du container :

```
docker commit 6982a9948422 webdown404/ping
```

Notes :

- -m permet d'ajouter un message au commit :

```
docker commit -m "mon premier commit" 6982a9948422 webdown404/ping
```

- -a permet de rajouter l'auteur d'un commit :

```
docker commit -a "Quentin BABAULT" 6982a9948422 webdown404/ping
```

Imaginons que nous souhaitons modifier webdown404/ping en pigant directement "www.google.com" :

```
docker run webdown404/ping ping www.google.com  
docker ps -l
```

Affiche l'ID "45af55fffzf..."

Une image est en lecture seule, donc si l'on souhaite apporter des modifications à notre image, il faut ajouter un tag (ici le tag « google ») :

```
docker commit 45af55fffzf webdown404/ping:google
```



Autres commandes de gestion des conteneurs

create Créer un nouveau conteneur
rm Supprimer un ou plusieurs conteneurs
rename Renommer un conteneur
start Démarrer un ou plusieurs conteneurs arrêtés
stop Arrête un ou plusieurs conteneurs en cours d'exécution
restart Redémarrer un ou plusieurs conteneurs
exec Exécuter une commande dans un conteneur en cours d'exécution
kill Tuer un ou plusieurs conteneurs en cours d'exécution
pause Pause tous les processus dans un ou plusieurs conteneurs
attach Attachez des flux d'entrée, de sortie et d'erreur locaux standard à un conteneur en cours d'exécution
cp Copier des fichiers / dossiers entre un conteneur et le système de fichiers local
unpause Mettre en pause tous les processus dans un ou plusieurs conteneurs
update mettre à jour la configuration d'un ou plusieurs conteneurs
wait Attendez jusqu'à ce qu'un ou plusieurs conteneurs s'arrêtent, puis imprimez leurs codes de sortie
ps Liste des conteneurs



3.4 Interface graphique

3.4.1 Shipyard

Shipyard est une interface utilisateur graphique conviviale pour gérer vos conteneurs. Shipyard utilise RethinkDB pour sa base de données. D'abord on va lancer un conteneur de RethinkDB :

```
docker run -ti -d --restart=always --name shipyard-rethinkdb rethinkdb
```

Voici cette commande en version multiligne :

```
docker run \  
-ti \  
-d \  
--restart=always \  
--name shipyard-rethinkdb \  
rethinkdb
```

Voici le retour attendu :

```
root@alexdocker01:~# docker run \  
> -ti \  
> -d \  
> --restart=always \  
> --name shipyard-rethinkdb \  
> rethinkdb  
Unable to find image 'rethinkdb:latest' locally  
latest: Pulling from library/rethinkdb  
e7a7e6031030: Pull complete  
0ca7c739ad6e: Pull complete  
c3237801dc54: Pull complete  
5d6127e7f6f6: Pull complete  
68af5fd8cef9: Pull complete  
Digest: sha256:7af2b5a808176e9a83c6aae97f5c6e3437cef0d359e62693edda1cff5d364af4  
Status: Downloaded newer image for rethinkdb:latest  
8f6352bfd2424c374f0cfa13c6671c966d4ff9c969e9836eee88b563de9798b8
```

Comme vous pouvez le constater Docker n'a pas trouvé l'image en local. Celle-ci a donc été téléchargée depuis le dépôt de dockerhub de façon automatique.



Proxy

Par default, Docker écoute seulement des "sockets". On va utiliser un conteneur proxy. Il s'agit d'un conteneur qui transmet simplement les requêtes TCP au socket Unix sur lequel Docker écoute.

```
#docker run \  
-ti \  
-d \  
-p 2375:2375 \  
--hostname=$HOSTNAME \  
--restart=always \  
--name shipyard-proxy \  
-v /var/run/docker.sock:/var/run/docker.sock \  
-e PORT=2375 \  
shipyard/docker-proxy:latest
```

Voici le retour attendu :

```
root@alexdocker01:~# docker run \  
> -ti \  
> -d \  
> -p 2375:2375 \  
> --hostname=$HOSTNAME \  
> --restart=always \  
> --name shipyard-proxy \  
> -v /var/run/docker.sock:/var/run/docker.sock \  
> -e PORT=2375 \  
> shipyard/docker-proxy:latest  
  
Unable to find image 'shipyard/docker-proxy:latest' locally  
latest: Pulling from shipyard/docker-proxy  
Image docker.io/shipyard/docker-proxy:latest uses outdated schema1 manifest  
format. Please upgrade to a schema2 image for better future compatibility. More  
information at https://docs.docker.com/registry/spec/deprecated-schema-v1/  
8f4ec95ceaae: Pull complete  
ac77a345f217: Pull complete  
43039e3ef672: Pull complete  
a3ed95caeb02: Pull complete  
Digest: sha256:da6bbd1a145581a940d44940cce0f43705d7f8ec552a4e97e77104ec1b6dc3d1  
Status: Downloaded newer image for shipyard/docker-proxy:latest  
42f6d4ac837b73bdf4c4b03cb2addec6112ec62e68ad1d1d00efe025a329e698
```



Contrôleur

Cela lance le contrôleur de Shipyard :

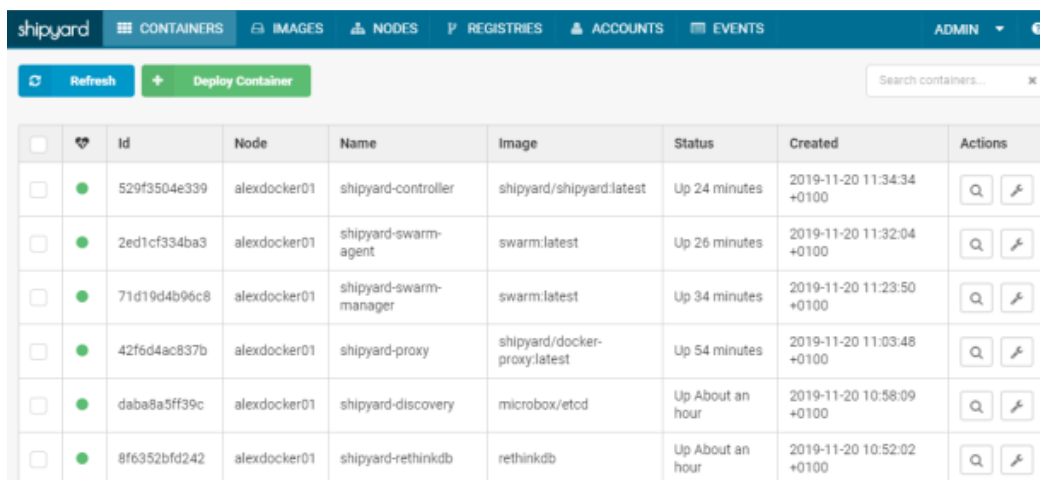
```
#docker run \
-ti \
-d \
--restart=always \
--name shipyard-controller \
--link shipyard-rethinkdb:rethinkdb \
--link shipyard-swarm-manager:swarm \
-p 8080:8080 \
shipyard/shipyard:latest \
server \
-d tcp://swarm:3375
```

Voici le retour :

```
root@docker01:~# docker run \
> -ti \
> -d \
> --restart=always \
> --name shipyard-controller \
> --link shipyard-rethinkdb:rethinkdb \
> --link shipyard-swarm-manager:swarm \
> -p 8080:8080 \
> shipyard/shipyard:latest \
> server \
> -d tcp://swarm:3375

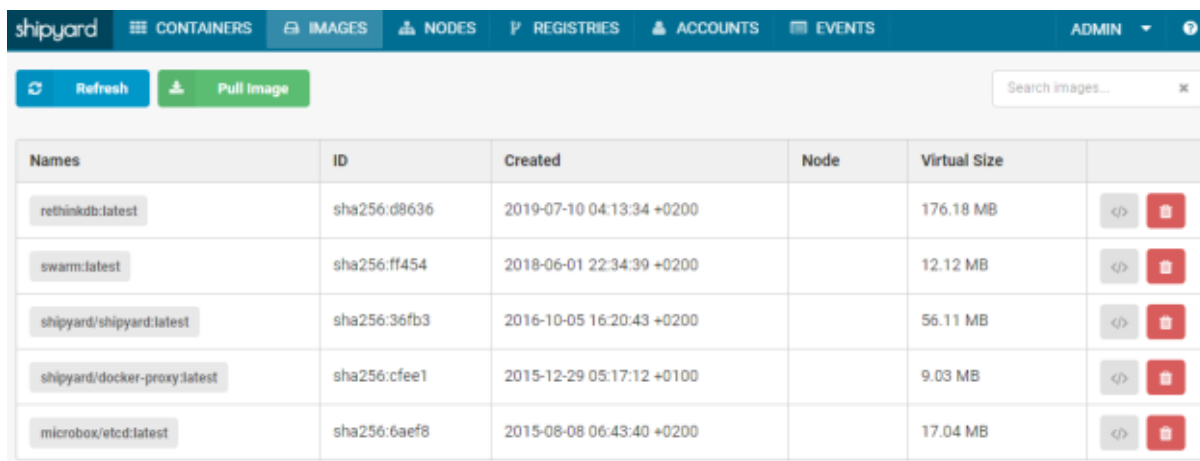
Unable to find image 'shipyard/shipyard:latest' locally
latest: Pulling from shipyard/shipyard
cb5507795515: Pull complete
fd711d385b34: Pull complete
9f2a509de079: Pull complete
a5251eb716bb: Pull complete
Digest: sha256:5f065362680fa4565dd150c8da3edd09b79a7a3010d3ceef20093c2a879187e0
Status: Downloaded newer image for shipyard/shipyard:latest
529f3504e339f5c1b496f813eba280ba7af26aedf38b44f7b855585a58438353
```











L'installation est maintenant finie. Vous pouvez vous connectez via :
[http:// \[IP-of-host\]:8080](http://[IP-of-host]:8080)



		Id	Node	Name	Image	Status	Created	Actions
<input type="checkbox"/>		529f3504e339	alexdocker01	shipyard-controller	shipyard/shipyard:latest	Up 24 minutes	2019-11-20 11:34:34 +0100	
<input type="checkbox"/>		2ed1cf334ba3	alexdocker01	shipyard-swarm-agent	swarm:latest	Up 26 minutes	2019-11-20 11:32:04 +0100	
<input type="checkbox"/>		71d19d4b96c8	alexdocker01	shipyard-swarm-manager	swarm:latest	Up 34 minutes	2019-11-20 11:23:50 +0100	
<input type="checkbox"/>		42f6d4ac837b	alexdocker01	shipyard-proxy	shipyard/docker-proxy:latest	Up 54 minutes	2019-11-20 11:03:48 +0100	
<input type="checkbox"/>		daba8a5ff39c	alexdocker01	shipyard-discovery	microbox/etcd	Up About an hour	2019-11-20 10:58:09 +0100	
<input type="checkbox"/>		8f6352bfd242	alexdocker01	shipyard-rethinkdb	rethinkdb	Up About an hour	2019-11-20 10:52:02 +0100	





Names	ID	Created	Node	Virtual Size	
rethinkdb:latest	sha256:d8636	2019-07-10 04:13:34 +0200		176.18 MB	 
swarm:latest	sha256:ff454	2018-06-01 22:34:39 +0200		12.12 MB	 
shipyard/shipyard:latest	sha256:36fb3	2016-10-05 16:20:43 +0200		56.11 MB	 
shipyard/docker-proxy:latest	sha256:cfee1	2015-12-29 05:17:12 +0100		9.03 MB	 
microbox/etc:latest	sha256:6aef8	2015-08-08 06:43:40 +0200		17.04 MB	 

Nous ne détaillerons pas l'utilisation de Shipyard. Les documentations ne manquent pas sur la toile.

Il faut cependant retenir les éléments suivants :

- Nous avons privilégié une répartition des services par conteneur. Mais nous aurions pu opter pour une simplicité d'installation en effectuant un « docker pull » de l'image complète comme ceci :

```
curl -sSL https://shipyard-project.com/deploy | bash -s
```

Nous avons installé Shipyard à titre d'exemple mais celui-ci s'appuie sur SWARM qui sera désactivé lorsque nous installerons Kubernetes.



3.4.2 Portainer

Portainer est une interface utilisateur de gestion légère à source ouverte pour les environnements Docker.

Il fonctionne au-dessus de l'API Docker et fournit une présentation détaillée de Docker, les fonctionnalités incluent la capacité de gérer des conteneurs, des images, des réseaux et des volumes.

Pour l'installation de portainer, nous allons grâce à Docker fraîchement installé, télécharger le conteneur :

```
sudo docker pull portainer/portainer:linux-arm
```

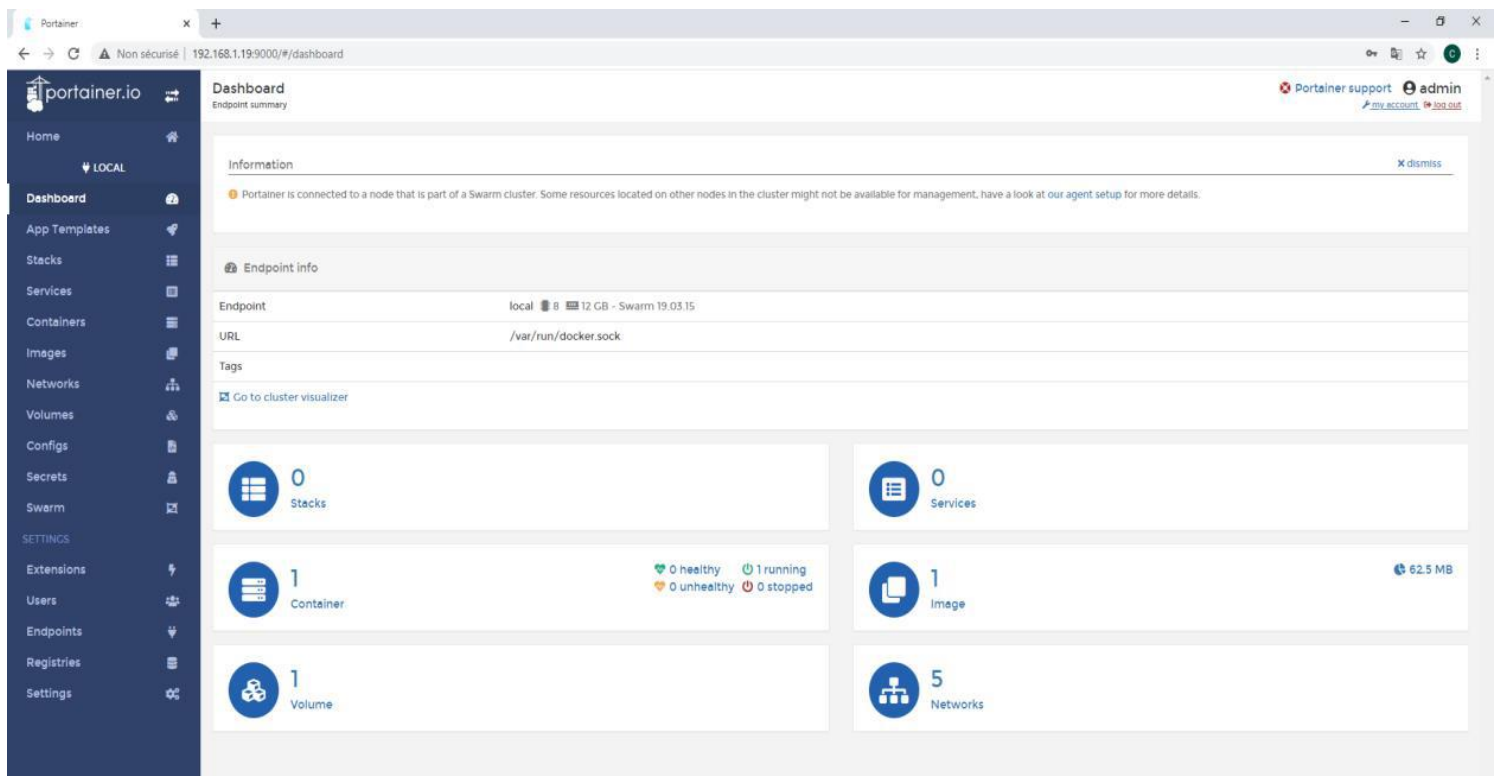
Puis nous allons démarrer le conteneur :

```
sudo docker run -d -p 9000:9000 -v /var/run/docker.sock:/var/run/docker.sock -v portainer_data:/data portainer/portainer
```

Vous pouvez maintenant accéder à l'interface Web de portainer via l'adresse suivante

http://IP_hôte:9000

Il ne manque plus qu'à créer votre mot de passe avec le login admin (il devra contenir 8 caractères minimum).



3.5 Gestion des conteneurs

3.5.1 Images Docker

Les conteneurs Docker sont construits à partir d'images Docker. Par défaut, Docker extrait ces images du dépôt « Docker Hub » (voir <https://hub.docker.com>). Ce registre Docker est géré par Docker (la société à l'origine du projet Docker). Tout le monde peut héberger leurs images Docker sur Docker Hub. Les images des applications et les distributions Linux dont vous aurez besoin y seront hébergées.

Pour vérifier si vous pouvez accéder aux images et les télécharger depuis Docker Hub, tapez :

```
docker run hello-world
```

La sortie indiquera que Docker fonctionne correctement :

```
Unable to find image 'hello-world:latest' locally latest: Pulling from
library/hello-world 9bb5a5d4561a: Pull complete Digest:
sha256:3e1764d0f546ceac4565547df2ac4907fe46f007ea229fd7ef2718514bcec35d Status:
Downloaded newer image for hello-world:latest Hello from Docker! This message
shows that your installation appears to be working correctly. ...
```

Docker était initialement incapable de 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 a été exécutée, affichant 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, tapez :

```
docker search ubuntu
```



La chaîne de caractères recherchée « ubuntu » est alors trouvée dans plusieurs résultats

ubuntu			Ubuntu is a Debian-
based Linux operating sys...	12685	[OK]	
dorowu/ubuntu-desktop-lxde-vnc			Docker image to
provide HTML5 VNC interface ...	560	[OK]	
websphere-liberty			WebSphere Liberty
multi-architecture images ...	280	[OK]	
rastasheep/ubuntu-sshd			Dockerized SSH
service, built on top of offi...	255	[OK]	
consol/ubuntu-xfce-vnc			Ubuntu container with
"headless" VNC session...	241	[OK]	
ubuntu-upstart			DEPRECATED, as is
Upstart (find other proces...	113	[OK]	
1and1internet/ubuntu-16-nginx-php-phpmyadmin-mysql-5			ubuntu-16-nginx-php-
phpmyadmin-mysql-5	50	[OK]	
ubuntu-debootstrap			DEPRECATED; use
"ubuntu" instead	44	[OK]	
i386/ubuntu			Ubuntu is a Debian-
based Linux operating sys...	25		
solita/ubuntu-systemd			Ubuntu + systemd
24		[OK]	
nuagebec/ubuntu			Simple always updated
Ubuntu docker images w...	24		
		[OK]	
fnndsc/ubuntu-python3			A slim Ubuntu-based
Python3 image	24		
		[OK]	
1and1internet/ubuntu-16-apache-php-5.6			ubuntu-16-apache-php-
5.6	14		
		[OK]	
1and1internet/ubuntu-16-apache-php-7.0			ubuntu-16-apache-php-
7.0	13		
		[OK]	
1and1internet/ubuntu-16-nginx-php-phpmyadmin-mariadb-10			ubuntu-16-nginx-php-
phpmyadmin-mariadb-10	11		
		[OK]	

Le « [OK] » (Dans la colonne « OFFICIELLE ») indique une image créée et prise en charge 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 ubuntu officielle sur votre ordinateur :

```
docker pull ubuntu
```

Vous verrez la sortie suivante :

```
Using default tag: latest latest: Pulling from library/ubuntu
6b98dfc16071: Pull complete
4001a1209541: Pull complete
6319fc68c576: Pull complete
b24603670dc3: Pull complete
97f170c87c6f: Pull complete
Digest: sha256:5f4bdc3467537cbb5e563e80db2c3ec95d548a9145d64453b06939c4592d67b6d
Status: Downloaded newer image for ubuntu:latest
```



Une fois qu'une image a été téléchargée, vous pouvez exécuter un conteneur à l'aide de l'image téléchargée avec la sous-commande « run ». Comme vous l'avez vu dans l'exemple hello-world, si une image n'a pas été téléchargée lorsqu'elle docker est exécutée avec la sous-commande « run », le client Docker télécharge d'abord l'image, puis exécute un conteneur qui l'utilise.

Pour voir les images téléchargées sur votre ordinateur, tapez :

```
docker images
```

La sortie devrait ressembler à ce qui suit :

REPOSITORY	TAG	IMAGE ID	CREATED	SIZE
ubuntu	latest	1318b700e415	4 weeks ago	72.8MB
hello-world	latest	d1165f221234	5 months ago	13.3kB

Les images que vous utilisez pour exécuter des conteneurs peuvent être modifiées et utilisés pour générer de nouvelles images, qui peut ensuite être téléchargé (poussé est le terme technique) à Docker Hub ou d'autres registres Docker.

Regardons comment exécuter des conteneurs plus en détail.



3.5.2 Conteneurs

Le conteneur hello-world que vous avez exécuté à l'étape précédente est un exemple de conteneur qui s'exécute et se ferme 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 plus faciles à utiliser.

À titre d'exemple, exécutons un conteneur à l'aide de 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
```

Votre invite de commande doit changer pour refléter le fait que vous travaillez maintenant à l'intérieur du conteneur et que vous devez prendre cette forme : `root@d9b100f2f636:/#`

Notez l'ID du conteneur dans l'invite de commande. Dans cet exemple, c'est le cas `d9b100f2f636`. Vous aurez besoin de cet ID de conteneur ultérieurement pour identifier le conteneur lorsque vous souhaitez le supprimer.

Vous pouvez maintenant exécuter n'importe quelle commande à l'intérieur du conteneur. Par exemple, mettons à jour la base de données de packages dans le conteneur. Vous n'avez pas besoin de préfixer une commande avec `sudo`, car vous travaillez dans le conteneur en tant qu'utilisateur `root` :

```
apt update
```

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

```
apt install nodejs
```

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

```
node -v
```

Vous verrez le numéro de version affiché dans votre terminal :

```
v8.10.0
```

Toute modification apportée à l'intérieur du conteneur ne s'applique qu'à ce conteneur.

Pour quitter le conteneur, tapez « `exit` » à l'invite.

Regardons maintenant la gestion des conteneurs sur notre système.



3.5.3 Gérer les conteneurs et images

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

```
docker ps
```

Vous verrez une sortie similaire à la suivante :

```
CONTAINER ID IMAGE COMMAND CREATED
```

Dans ce tutoriel, vous avez démarré deux conteneurs ; un de l'image hello-world et un autre de l'image ubuntu. Les deux conteneurs ne fonctionnent plus, mais ils existent toujours sur votre système.

Pour afficher tous les conteneurs - actifs et inactifs, exécutez docker ps le commutateur -a :

```
docker ps -a
```

Vous verrez une sortie similaire à ceci :

CONTAINER ID	IMAGE	COMMAND	CREATED	STATUS
3c4b70eeea3d	hello-world	"/hello"	11 seconds ago	Exited (0) 11 seconds ago
6aa4d7c3a208	ubuntu	"bash"	18 seconds ago	Exited (0) 18 seconds ago

Pour afficher le dernier conteneur que vous avez créé, passez le commutateur -l :

```
docker ps -l
```

CONTAINER ID	IMAGE	COMMAND	CREATED	STATUS
3c4b70eeea3d	hello-world	"/hello"	About a minute ago	Exited (0) About a minute ago



Pour démarrer un conteneur arrêté, utilisez `docker start`, suivi de l'ID du conteneur ou du nom du conteneur. Commençons le conteneur basé sur Ubuntu avec l'ID de `d9b100f2f636`:

```
docker start d9b100f2f636
```

Le conteneur démarrera et vous pourrez voir son statut `docker ps`.

Pour arrêter un conteneur en cours d'exécution, utilisez `docker stop`, suivi de l'ID ou du nom du conteneur. Cette fois, nous utiliserons le nom que Docker a attribué au conteneur, à savoir « `sharp_volhard` » :

```
docker stop sharp_volhard
```

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

```
docker rm festive_williams
```

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

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



Validation des modifications

Lorsque vous démarrez une image Docker, vous pouvez créer, modifier et supprimer des fichiers comme vous pouvez le faire 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'avez détruit avec la commande `docker rm`, les modifications seront définitivement perdues.

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 exécute 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 ultérieurement.

Puis validez les modifications sur 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_image_name
```

Le commutateur `-m` est pour le **message** de validation qui vous aide, vous et les autres, à savoir quelles modifications vous avez apportées, tandis que `-a` est utilisé pour spécifier l'auteur.

Le **container_id** est celui que vous avez noté précédemment dans le didacticiel lorsque vous avez démarré la session interactive Docker. À moins que vous ayez créé des référentiels supplémentaires sur Docker Hub, il s'agit généralement de votre nom d'utilisateur Docker Hub (c'est le même que votre repository).

Par exemple, pour l'utilisateur `afalzon`, avec l'ID du conteneur de `d9b100f2f636`, la commande serait : `docker commit -m "added Node.js" -a "afalzon" d9b100f2f636 afalzon/ubuntu-nodejs`. Lorsque vous validez une image, la nouvelle image est enregistrée localement sur votre ordinateur. Plus loin dans ce didacticiel, vous apprendrez comment transférer une image vers un registre Docker comme Docker Hub pour que d'autres puissent y accéder.

La liste des images Docker à nouveau affichera la nouvelle image, ainsi que l'ancienne image dérivée :

```
docker images
```

Vous verrez le résultat suivant :

REPOSITORY	TAG	IMAGE ID	CREATED	SIZE
ubuntu-nodejs	latest	7c1f35226ca6	7 seconds ago	179MB
ubuntu	latest	1318b700e415	4 weeks ago	72.8MB
hello-world	latest	d1165f221234	5 months ago	13.3kB

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

Vous pouvez également créer des images à partir d'un Docker file fichier, ce 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 les autres afin qu'ils puissent en créer des conteneurs.



3.5.4 Exporter une image dans le Docker hub

La prochaine étape logique après la création d'une nouvelle image à partir d'une image existante consiste à la partager avec quelques-uns de vos amis, le monde entier sur Docker Hub ou tout autre registre Docker auquel vous avez accès. Pour transférer une image vers Docker Hub ou tout autre registre Docker, vous devez y avoir un compte.

Cette section vous montre comment transférer une image Docker vers Docker Hub. Pour apprendre à créer votre propre registre Docker privé, consultez Comment configurer un registre Docker privé.

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

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

Vous serez invité à vous authentifier à l'aide de votre mot de passe Docker Hub. Si vous avez spécifié le mot de passe correct, l'authentification doit aboutir. Ensuite, vous pouvez pousser votre propre image en utilisant :

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

Pour pousser l'image ubuntu-nodejs dans le référentiel afalzon , la commande serait:

```
docker push afalzon/ubuntu-nodejs
```

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

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

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

Nouvelle liste d'images Docker sur Docker Hub

Si une tentative de poussée entraîne une erreur de ce type, vous ne vous êtes probablement pas connecté :

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

Connectez-vous avec le login docker et répétez la tentative de poussée. Ensuite, vérifiez qu'il existe sur votre page de référentiel Docker Hub.

Vous pouvez maintenant utiliser l'image pour la transférer sur une nouvelle machine et l'utiliser pour exécuter un nouveau conteneur.

```
docker pull afalzon/ubuntu-node
```



3.5.5 Communication entre conteneurs

Pour pouvoir faire communiquer deux containers entre eux il existe deux possibilités :

1. Lier les containers via les ports réseaux
2. Lier les containers via une variable d'environnement (l'option `--link`)

L'option `--link` est généralement la meilleure solution. Elle s'utilise de la manière suivante : `docker run -d --name db training/postgres docker run -d -P -name web --link db:db training/webapp python app.py`

Il est également possible d'utiliser l'ID à la place du nom.

3.5.6 Monter un répertoire hôte dans un conteneur

Pour monter un répertoire hôte dans un container, il faut utiliser l'option `-v` :

```
docker run -P -v /Users/webdown/www/sandbox/:/www python python /www/app.py
```

`/Users/webdown/www/sandbox` correspond au chemin absolu de la machine hôte qui ici n'est pas le PC mais Boot2Docker. Virtualbox partage par défaut le dossier `"/Users"` en le montant dans le dossier `"/Users"` de Boot2Docker.

`:/www` correspond à l'emplacement dans lequel sera monté le dossier partagé dans le container

Il est également possible de limiter l'accès à un dossier en lecture seule en rajoutant `:ro` :
`/Users/webdown/www/sandbox/:/www:ro`



3.5.7 Conteneur de données

Si on souhaite persister des données utilisables entre les différents containers, la meilleure solution est d'utiliser un container de données.

Pour créer un container de données

```
docker create -v /dbdata --name dbdata training/postgres /bin/true
```

Pour monter le volume /dbdata du container de données dans plusieurs autres containers :

```
docker run -d --volumes-from dbdata --name db1 training/postgres  
docker run -d --volumes-from dbdata --name db2 training/postgres
```

Si on souhaite supprimer un container de données, il faut utiliser l'option -v sinon on risque de se retrouver avec des volumes fantômes (Dangling volumes)

Un avantage d'utiliser un container de données est pouvoir faire facilement des backup/restaurations de données.

Pour réaliser une sauvegarde :

```
docker run --volumes-from dbdata -v $(pwd):/backup ubuntu tar cvf  
/backup/backup.tar /dbdata
```

Pour créer un nouveau container de données et y restaurer les données du container de données précédent :

```
docker run -v /dbdata --name dbdata2 ubuntu /bin/bash  
docker run --volumes-from dbdata2 -v /backup busybox tar xvf /backup/backup.tar
```



3.5.8 Swarm

SWARM est l'orquestrateur natif de Docker. Bien que le monde entier se dirige vers l'utilisation massive de Kubernetes (l'orquestrateur développé par Google). Nous allons découvrir l'utilisation de SWARM au passage.

Par default, Docker écoute seulement des "sockets". On va utiliser un conteneur proxy. Il s'agit d'un conteneur qui transmet simplement les requêtes TCP au socket Unix sur lequel Docker écoute.

Pour l'activation de SWARM, nous devons utiliser une clé externe du conteneur SWARM. Pour cet exemple, nous utiliserons etcd :

```
docker run \  
-ti \  
-d \  
-p 4001:4001 \  
-p 7001:7001 \  
--restart=always \  
--name shipyard-discovery \  
microbox/etcd -name discovery
```

Voici le retour attendu :

```
root@docker01:~# docker run \  
> -ti \  
> -d \  
> -p 4001:4001 \  
> -p 7001:7001 \  
> --restart=always \  
> --name shipyard-discovery \  
> microbox/etcd -name discovery  
Unable to find image 'microbox/etcd:latest' locally  
latest: Pulling from microbox/etcd  
Image docker.io/microbox/etcd:latest uses outdated schema1 manifest format.  
Please upgrade to a schema2 image for better future compatibility. More  
information at https://docs.docker.com/registry/spec/deprecated-schema-v1/  
8ded6e8ab3fd: Pull complete  
bf8f85223d7a: Pull complete  
a3ed95caeb02: Pull complete  
Digest: sha256:941fd46b4eab265c65da9bfbf33397b853a7cef6c16df93a1e3fea7b4e47fc90  
Status: Downloaded newer image for microbox/etcd:latest  
daba8a5ff39cc76a8282cf7d564a460bfb5de7231f7932ec284d0a800fc593e0
```



Swarm manager

Voici maintenant la création d'un conteneur SWARM préconfiguré :

```
#docker run \  
-ti \  
-d \  
--restart=always \  
--name shipyard-swarm-manager \  
swarm:latest \  
manage --host tcp://0.0.0.0:3375 etcd://<IP-OF-HOST>:4001
```

Il faut mettre l'IP de votre machine en lieu et place de <IP-OF-HOST>.

Voici le retour :

```
root@alexdocker01:~# docker run \  
> -ti \  
> -d \  
> --restart=always \  
> --name shipyard-swarm-manager \  
> swarm:latest \  
> manage --host tcp://0.0.0.0:3375 etcd://192.168.8.100:4001  
Unable to find image 'swarm:latest' locally  
latest: Pulling from library/swarm  
d85c18077b82: Pull complete  
1e6bb16f8cb1: Pull complete  
85bac13497d7: Pull complete  
Digest: sha256:b866583a3b8791bcd705b7bc0fd94c66b695a1a2dbaeb5f59ed29940e5015dc8  
Status: Downloaded newer image for swarm:latest  
71d19d4b96c8d65e704429a8953abc546792092510022e3407da4fead63fe4fe
```



Swarm Agent

Ce conteneur exécute un agent SWARM qui permet au noeud de planifier les conteneurs.

```
#docker run \  
-ti \  
-d \  
--restart=always \  
--name shipyard-swarm-agent \  
swarm:latest \  
join --addr <ip-of-host>:2375 etcd://<ip-of-host>:4001
```

Il faut mettre l'IP de votre machine en lieu et place de <ip-of-host>.

Voici le retour :

```
root@alexdocker01:~# docker run \  
> -ti \  
> -d \  
> --restart=always \  
> --name shipyard-swarm-agent \  
> swarm:latest \  
> join --addr 192.168.8.100:2375 etcd://192.168.8.100:4001  
2ed1cf334ba3d7f76a30e8d51db491c3d70c67d4c0da4b3b5da29b3802e661af
```



3.5.9 Kubernetes

3.6 Installation et configuration de Kubernetes

3.6.1 Introduction

Kubernetes est une plateforme open source développée par Google pour la gestion des applications conteneurisées. Il vous permet de gérer, de mettre à l'échelle et de déployer automatiquement vos applications conteneurisées dans l'environnement en cluster.

Avec Kubernetes, vous pouvez :

- Orchestrer des conteneurs sur plusieurs hôtes,
- Mettre à l'échelle les applications conteneurisées avec toutes les ressources à la volée
- Et disposer d'un environnement de gestion de conteneur centralisé.

A titre d'exemple voici comment installer et configurer Kubernetes sur Ubuntu 18.04. Nous utiliserons 3 serveurs Ubuntu 18.04LTS avec les privilèges « root » :

- « alexdockerk8smaster » avec l'IP 192.168.8.102 (Maître Kubernetes)
- « alexdocker01 » avec l'IP 192.168.8.100 (noeuds de travail Kubernetes)
- « alexdocker02 » avec l'IP 192.168.8.101 (noeuds de travail Kubernetes)

ATTENTION

Si vous souhaitez dupliquer une VM pour éviter des installations multiples de Ubuntu 18.04 server il vous faudra changer les noms d'hôte des serveurs pour que les « Workers » puis être visible dans le « master »

Pour cela éditer le fichier « /etc/cloud/cloud.cfg

```
root@dockerk8smaster:~# nano /etc/cloud/cloud.cfg
```

Puis modifier la ligne « preserve_hostname: false » en la remplaçant par « preserve_hostname: true

```
# This will cause the set+update hostname module to not operate (if true)
preserve_hostname: true
```

Renommer le nom d'hôte puis redémarrer le serveur :

```
root@dockerk8smaster:~# echo alexdockerk8smaster > /etc/hostname
root@dockerk8smaster:~# reboot
```

Voici les étapes à réaliser :

- Installation de Kubeadm
- Configuration des hôtes
- Installation de Docker
- Désactivation de la SWAP
- Installation des paquets Kubeadm
- Initialisation du cluster Kubernetes



- Ajout de noeuds de travail au cluster Kubernetes
- Réalisation de tests

3.6.2 Préparation des hôtes

Dans cette première étape, nous allons préparer ces 3 serveurs pour l'installation de Kubernetes. Exécutez donc toutes les commandes sur les noeuds maître et de travail.

Nous allons préparer tous les serveurs pour l'installation de Kubernetes en modifiant la configuration existante sur les serveurs et en installant également certains packages, y compris docker et kubernetes.

Editez le fichier hosts sur tous les serveurs à l'aide de l'éditeur nano ou vim :

```
sudo vim /etc/hosts
```

Collez la configuration des hôtes ci-dessous :

```
192.168.8.102 dockerk8smaster
192.168.8.100 docker01
192.168.8.101 docker02
```

Sauvegarder et quitter.

Maintenant testez le ping sur tous les serveurs nom d'hôte :

```
ping -c 3 alexdockerk8smaster
ping -c 3 alexdocker01
ping -c 3 alexdocker02
```

Assurez-vous que toutes les adresses IP sont résolues en tant que nom d'hôte.

```
root@dockerk8smaster:~# ping -c 3 dockerk8smaster
PING localhost.localdomain (127.0.0.1) 56(84) bytes of data.
64 bytes from localhost.localdomain (127.0.0.1): icmp_seq=1 ttl=64 time=0.067 ms
64 bytes from localhost.localdomain (127.0.0.1): icmp_seq=2 ttl=64 time=0.038 ms
64 bytes from localhost.localdomain (127.0.0.1): icmp_seq=3 ttl=64 time=0.045 ms
--- localhost.localdomain ping statistics ---
3 packets transmitted, 3 received, 0% packet loss, time 2031ms
rtt min/avg/max/mdev = 0.038/0.050/0.067/0.012 ms
root@ dockerk8smaster:~# ping -c 3 docker01
PING docker01 (192.168.8.100) 56(84) bytes of data.
64 bytes from docker01 (192.168.8.100): icmp_seq=1 ttl=64 time=0.514 ms
64 bytes from docker01 (192.168.8.100): icmp_seq=2 ttl=64 time=0.290 ms
64 bytes from docker01 (192.168.8.100): icmp_seq=3 ttl=64 time=0.411 ms
--- alexdocker01 ping statistics ---
3 packets transmitted, 3 received, 0% packet loss, time 2044ms
rtt min/avg/max/mdev = 0.290/0.405/0.514/0.091 ms
root@ dockerk8smaster:~# ping -c 3 docker02
PING docker02 (192.168.8.101) 56(84) bytes of data.
64 bytes from alexdocker02 (192.168.8.101): icmp_seq=1 ttl=64 time=1.92 ms
64 bytes from alexdocker02 (192.168.8.101): icmp_seq=2 ttl=64 time=0.366 ms
64 bytes from alexdocker02 (192.168.8.101): icmp_seq=3 ttl=64 time=0.323 ms
--- docker02 ping statistics ---
rtt min/avg/max/mdev = 0.323/0.871/1.925/0.745 ms
```



Désactiver la Swap

Afin de configurer les serveurs Linux Kubernetes, nous devons désactiver le SWAP. Vérifiez la liste de swap et désactivez-la.

```
sudo swapon -s  
sudo swapoff -a
```

Pour désactiver le SWAP de manière permanente, nous devons éditer le fichier '/etc/fstab'.

```
sudo vim /etc/fstab
```

Faites un commentaire sur le type de partition SWAP.

```
#/dev/mapper/hakase--labs--vg-swap_1 none swap sw 0 0
```

Enregistrez et quittez, puis redémarrez le système.

```
sudo reboot
```

3.6.3 Installation des Kubeadm

Dans ce tutoriel, nous allons utiliser les packages Kubeadm pour configurer le cluster Kubernetes. Nous installerons ceux-ci à partir du référentiel officiel Kubernetes.

Installez « apt-transport-https ».

```
sudo apt install -y apt-transport-https
```

Ajoutez la clé APT de Kubernetes

```
curl -s https://packages.cloud.google.com/apt/doc/apt-key.gpg | sudo apt-key  
add -
```

Et ajoutez le référentiel Kubernetes en créant un nouveau fichier kubernetes.list dans le répertoire « /etc/apt/sources.list.d ».

```
cd /etc/apt/ sudo vim sources.list.d/kubernetes.list
```

Coller le référentiel Kubernetes ci-dessous.

```
deb http://apt.kubernetes.io/ kubernetes-xenial main
```

Maintenant, mettez à jour le référentiel et installez les paquets kubeadm en utilisant les commandes « apt » ci-dessous.

```
sudo apt update sudo apt install -y kubeadm kubelet kubect1
```

Attendez l'installation des packages kubeadm.

```
root@dockerk8smaster:~# sudo apt install -y kubeadm kubelet kubect1  
Reading package lists... Done  
Building dependency tree Reading state information... Done  
kubeadm is already the newest version (1.16.3-00).  
kubect1 is already the newest version (1.16.3-00).  
kubelet is already the newest version (1.16.3-00).  
0 upgraded, 0 newly installed, 0 to remove and 3 not upgraded.
```



3.6.4 Initialisation du cluster

Dans cette étape, nous allons initialiser Kubernetes sur le noeud 'dockerk8smaster'.

ATTENTION

Exécutez toutes les commandes de cette étape uniquement sur le serveur 'alexdockerk8smaster'.

Initialisez le cluster Kubernetes à l'aide de la commande kubeadm ci-dessous.

```
sudo kubeadm init --pod-network-cidr=10.244.0.0/16 --apiserver-advertise-address=192.168.8.102 --kubernetes-version "1.16.0"
```

« --apiserver-advertise-address = » détermine l'adresse IP sur laquelle Kubernetes doit annoncer son serveur d'API.

« --pod-network-cidr = » spécifie la plage d'adresses IP du réseau de pod. Nous utilisons le réseau virtuel 'flannel'. Si vous souhaitez utiliser un autre réseau de pods tel que weave-net ou Calico, modifiez l'adresse IP de la plage.

RAPPEL

En cas de problème vous avez la possibilité de recommencer cette étape en réalisant au préalable un

```
Kubeadm reset
```

Puis effacer le repertoire \$HOME/.kube

```
mkdir -p $HOME/.kube
```



Une fois l'initialisation de Kubernetes terminée, vous obtiendrez le résultat indiqué ci-dessous.

```
root@alexdocker01:~# sudo kubeadm init --pod-network-cidr=10.244.0.0/16 --
apiserver-advertise-address=192.168.8.102 --kubernetes-version "1.16.0"
[init] Using Kubernetes version: v1.16.0
[preflight] Running pre-flight checks
[WARNING IsDockerSystemdCheck]: detected "cgroupfs" as the Docker cgroup
driver. The recommended driver is "systemd". Please follow the guide at https:/
/kubernetes.io/docs/setup/cri/
[WARNING SystemVerification]: this Docker version is not on the list of
validated versions: 19.03.5. Latest validated version: 18.09
[preflight] Pulling images required for setting up a Kubernetes cluster
[preflight] This might take a minute or two, depending on the speed of your
internet connection
[preflight] You can also perform this action in beforehand using 'kubeadm
config images pull'
[kubelet-start] Writing kubelet environment file with flags to file
"/var/lib/kubelet/kubeadm-flags.env"
[kubelet-start] Writing kubelet configuration to file
"/var/lib/kubelet/config.yaml"
[kubelet-start] Activating the kubelet service
[certs] Using certificateDir folder "/etc/kubernetes/pki"
[certs] Generating "ca" certificate and key
[certs] Generating "apiserver" certificate and key
[certs] apiserver serving cert is signed for DNS names [alexdocker01 kubernetes
kubernetes.default kubernetes.default.svc kubernetes.default.svc.cluster.local]
and IPs [10.96.0.1 192.168.8.102]
[certs] Generating "apiserver-kubelet-client" certificate and key
[certs] Generating "front-proxy-ca" certificate and key
[certs] Generating "front-proxy-client" certificate and key
[certs] Generating "etcd/ca" certificate and key
[certs] Generating "etcd/server" certificate and key
[certs] etcd/server serving cert is signed for DNS names [alexdocker01
localhost] and IPs [192.168.8.102 127.0.0.1 ::1]
[certs] Generating "etcd/peer" certificate and key
[certs] etcd/peer serving cert is signed for DNS names [alexdocker01 localhost]
and IPs [192.168.8.102 127.0.0.1 ::1]
[certs] Generating "etcd/healthcheck-client" certificate and key
[certs] Generating "apiserver-etcd-client" certificate and key
[certs] Generating "sa" key and public key
[kubeconfig] Using kubeconfig folder "/etc/kubernetes"
[kubeconfig] Writing "admin.conf" kubeconfig file
[kubeconfig] Writing "kubelet.conf" kubeconfig file
[kubeconfig] Writing "controller-manager.conf" kubeconfig file
[kubeconfig] Writing "scheduler.conf" kubeconfig file
[control-plane] Using manifest folder "/etc/kubernetes/manifests"
[control-plane] Creating static Pod manifest for "kube-apiserver"
[control-plane] Creating static Pod manifest for "kube-controller-manager"
[control-plane] Creating static Pod manifest for "kube-scheduler"
[etcd] Creating static Pod manifest for local etcd in
"/etc/kubernetes/manifests"
[wait-control-plane] Waiting for the kubelet to boot up the control plane as
static Pods from directory "/etc/kubernetes/manifests". This can take up to
4m0s
[apiclient] All control plane components are healthy after 38.010235 seconds
[upload-config] Storing the configuration used in ConfigMap "kubeadm-config" in
the "kube-system" Namespace
[kubelet] Creating a ConfigMap "kubelet-config-1.16" in namespace kube-system
with the configuration for the kubelets in the cluster
```



```
[upload-certs] Skipping phase. Please see --upload-certs
[mark-control-plane] Marking the node alexdocker01 as control-plane by adding
the label "node-role.kubernetes.io/master=''"
[mark-control-plane] Marking the node alexdocker01 as control-plane by adding
the taints [node-role.kubernetes.io/master:NoSchedule]
[bootstrap-token] Using token: p3jsv8.9svqnv6rgrgxjqww
[bootstrap-token] Configuring bootstrap tokens, cluster-info ConfigMap, RBAC
Roles
[bootstrap-token] configured RBAC rules to allow Node Bootstrap tokens to post
CSRs in order for nodes to get long term certificate credentials
[bootstrap-token] configured RBAC rules to allow the csrapprover controller
automatically approve CSRs from a Node Bootstrap Token
[bootstrap-token] configured RBAC rules to allow certificate rotation for all
node client certificates in the cluster
[bootstrap-token] Creating the "cluster-info" ConfigMap in the "kube-public"
namespace
[addons] Applied essential addon: CoreDNS
[addons] Applied essential addon: kube-proxy
Your Kubernetes control-plane has initialized successfully!
To start using your cluster, you need to run the following as a regular user:
mkdir -p $HOME/.kube
sudo cp -i /etc/kubernetes/admin.conf $HOME/.kube/config
sudo chown $(id -u):$(id -g) $HOME/.kube/config
You should now deploy a pod network to the cluster.
Run "kubectl apply -f [podnetwork].yaml" with one of the options listed at:
https://kubernetes.io/docs/concepts/cluster-administration/addons/
Then you can join any number of worker nodes by running the following on each
as root:
kubeadm join 192.168.8.102:6443 --token 9y7x5f.nrdvL5eilwsgyg9g \
--discovery-token-ca-cert-hash
sha256:8f3ce1b01c6c394d7297be499da9a52355cc350a3269e90326a7cbc124f3db80
```

ATTENTION : le Token ne sera plus retrouvable

Copiez donc la commande « kubeadm join » complète (avec le token) dans votre éditeur de texte.
La commande sera utilisée pour enregistrer de nouveaux noeuds de travail dans le cluster
Kubernetes.

Maintenant, pour utiliser Kubernetes, nous devons exécuter certaines commandes, comme indiqué
dans le résultat.



Créez un nouveau répertoire de configuration ".kube" et copiez le répertoire de configuration "admin.conf" à partir du répertoire "/etc/kubernetes".

```
mkdir -p $HOME/.kube
sudo cp -i /etc/kubernetes/admin.conf $HOME/.kube/config
sudo chown $(id -u):$(id -g) $HOME/.kube/config
```

Ensuite, déployez le réseau de flanelles sur le cluster Kubernetes à l'aide de la commande « kubectl ».

```
kubectl apply -f
https://raw.githubusercontent.com/coreos/flannel/master/Documentation/kube-
flannel.yml
```

```
root@alexdockerk8smaster:~# kubectl apply -f
https://raw.githubusercontent.com/coreos/flannel/master/Documentation/kube-
flannel.yml
podsecuritypolicy.policy/psp.flannel.unprivileged created
clusterrole.rbac.authorization.k8s.io/flannel created
clusterrolebinding.rbac.authorization.k8s.io/flannel created
serviceaccount/flannel created
configmap/kube-flannel-cfg created
daemonset.apps/kube-flannel-ds-amd64 created
daemonset.apps/kube-flannel-ds-arm64 created
daemonset.apps/kube-flannel-ds-arm created
daemonset.apps/kube-flannel-ds-ppc64le created
daemonset.apps/kube-flannel-ds-s390x created
Le réseau de flanelle a été déployé sur le cluster Kubernetes.
```

ATTENTION

Attendez une minute, avant de passer à la suite.

Vérifiez le noeud Kubernetes et les pods à l'aide des commandes ci-dessous.

```
kubectl get nodes
kubectl get pods --all-namespaces
```

Le noeud 'dockerk8smaster' s'exécute en tant que cluster 'maître' avec le statut 'prêt'

```
root@dockerk8smaster:~# kubectl get nodes
```

Tous les pods 'kube-system' nécessaires au cluster sont opérationnels.

```
root@dockerk8smaster:~# kubectl get nodes
NAME          STATUS  ROLES  AGE  VERSION
docker01 Ready master 72m v1.16.3
```

Tous les pods 'kube-system' nécessaires au cluster sont opérationnels.

```
root@dockerk8smaster:~# kubectl get pods --all-namespaces
NAMESPACE     NAME                                     READY STATUS RESTARTS AGE
kube-system   coredns-5644d7b6d9-cwf7j 1/1 Running 0       72m
kube-system   coredns-5644d7b6d9-dptjc 1/1 Running 0       72m
kube-system   etcd-alexdocker01 1/1 Running 0       72m
kube-system   kube-apiserver-alexdocker01 1/1 Running 0       71m
kube-system   kube-controller-manager-alexdocker01 1/1 Running 0       71m
kube-system   kube-flannel-ds-amd64-q1brv 1/1 Running 0       12m
kube-system   kube-proxy-pzblk 1/1 Running 0       72m
kube-system   kube-scheduler-alexdocker01 1/1 Running 0 72mTous Les pods 'kube-system'
nécessaires au cluster sont opérationnels.
```

L'initialisation et la configuration du maître de cluster Kubernetes sont terminées



Ajout des nœuds de travail au cluster

Dans cette étape, nous allons ajouter les deux travailleurs (ou slave ou worker) « docker01 » et « docker02 » au nœud Kubernetes.
Connectez-vous au serveur 'docker01' et exécutez la commande `kubeadm join` obtenue lors de l'initialisation du cluster.

```
kubeadm join 192.168.8.102:6443 --token 9y7x5f.nrdvl5eilwsgyg9g \  
--discovery-token-ca-cert-hash  
sha256:8f3ce1b01c6c394d7297be499da9a52355cc350a3269e90326a7cbc124f3db80
```

```
root@docker01:~# kubeadm join 192.168.8.102:6443 --token  
9y7x5f.nrdvl5eilwsgyg9g \  
> --discovery-token-ca-cert-hash sha256:  
8f3ce1b01c6c394d7297be499da9a52355cc350a3269e90326a7cbc124f3db80  
[preflight] Running pre-flight checks  
[WARNING IsDockerSystemdCheck]: detected "cgroupfs" as the Docker cgroup  
driver. The recommended driver is "systemd". Please follow the guide at  
https://kubernetes.io/docs/setup/cri/  
[WARNING SystemVerification]: this Docker version is not on the list of  
validated versions: 19.03.5. Latest validated version: 18.09  
[preflight] Reading configuration from the cluster...  
[preflight] FYI: You can look at this config file with 'kubectl -n kube-system  
get cm kubeadm-config -oyaml'  
[kubelet-start] Downloading configuration for the kubelet from the "kubelet-  
config-1.16" ConfigMap in the kube-system namespace  
[kubelet-start] Writing kubelet configuration to file  
"/var/lib/kubelet/config.yaml"  
[kubelet-start] Writing kubelet environment file with flags to file  
"/var/lib/kubelet/kubeadm-flags.env"  
[kubelet-start] Activating the kubelet service  
[kubelet-start] Waiting for the kubelet to perform the TLS Bootstrap...  
This node has joined the cluster:  
* Certificate signing request was sent to apiserver and a response was  
received.  
* The Kubelet was informed of the new secure connection details.  
Run 'kubectl get nodes' on the control-plane to see this node join the cluster.
```

Exécutez la même commande « `kubeadm join` » sur 'docker02' .



ATTENTION

Attendez quelques minutes avant de retourner sur le maître.

Revenez sur le maître de noeud « alexdockerk8smaster » et vérifiez l'état du noeud.

```
kubectl get nodes
```

Vous verrez que les noeuds de travail 'alexdocker01' et 'alexdocker02' font partie du cluster Kubernetes :

```
root@alexdockerk8smaster:~# kubectl get node
NAME STATUS ROLES AGE VERSION
alexdocker01 Ready <none> 91s v1.16.3
alexdocker02 Ready <none> 80s v1.16.3
alexdockerk8smaster Ready master 5m44s v1.16.3
```

RAPPEL

Les noms d'hôte des serveurs doivent être uniques pour que les nodes apparaissent dans le master.

Si ce n'est pas le cas reportez-vous à l'avertissement de la page 50.

Dans ce cas il faut réinitialiser le master comme suit :

```
root@alexdockerk8smaster:~# kubeadm reset
...
The reset process does not reset or clean up iptables rules or IPVS tables.
If you wish to reset iptables, you must do so manually by using the "iptables"
command.
If your cluster was setup to utilize IPVS, run ipvsadm --clear (or similar)
to reset your system's IPVS tables.
The reset process does not clean your kubeconfig files and you must remove them
manually.
Please, check the contents of the $HOME/.kube/config file.
```

Et recommencer à partir de l'étape Initialisation du cluster Kubernetes.

Réalisation des tests

Dans cette étape, nous allons déployer le serveur Web Nginx dans le cluster. Nous déploierons le serveur Web Nginx à l'aide du modèle YAML.

Effectuez les commandes suivantes uniquement sur le master.

Créez un nouveau répertoire nommé "nginx" et accédez à ce répertoire.

```
mkdir -p nginx/
cd nginx/
```

Créez maintenant le fichier Nginx Deployment YAML 'nginx-deployment.yaml' à l'aide de l'éditeur vim

```
sudo vim nginx-deployment.yaml
```



Coller les configurations ci-dessous.

```
apiVersion: apps/v1
kind: Deployment
metadata:
  name: nginx-deployment
  labels:
    app: nginx
spec:
  replicas: 3
  selector:
    matchLabels:
      app: nginx
  template:
    metadata:
      labels:
        app: nginx
    spec:
      containers:
      - name: nginx
        image: nginx:1.14.0
        ports:
        - containerPort: 80
```

Sauvegarder et quitter.

Créez maintenant le déploiement nommé "nginx-deployment" en exécutant la commande kubectl ci-dessous.

```
kubectl create -f nginx-deployment.yaml
root@alexdockerk8smaster:~/nginx# kubectl create -f nginx-deployment.yaml
deployment.apps/nginx-deployment created
```



Après avoir créé un nouveau "nginx-deployment", Vérifiez la liste des déploiements dans le cluster.

```
kubectl get deployments
kubectl describe deployment nginx-deployment
```

Voici le résultat :

```
root@alexdockerk8smaster:~/nginx# kubectl get deployments
NAME READY UP-TO-DATE AVAILABLE AGE
nginx-deployment 3/3 3 3 118s
root@alexdockerk8smaster:~/nginx# kubectl describe deployment nginx-deployment
Name: nginx-deployment
Namespace: default
CreationTimestamp: Fri, 22 Nov 2019 13:20:48 +0000
Labels: app=nginx
Annotations: deployment.kubernetes.io/revision: 1
Selector: app=nginx
Replicas: 3 desired | 3 updated | 3 total | 3 available | 0 unavailable
StrategyType: RollingUpdate
MinReadySeconds: 0
RollingUpdateStrategy: 25% max unavailable, 25% max surge
Pod Template:
  Labels: app=nginx
  Containers:
    nginx:
      Image: nginx:1.14.0
      Port: 80/TCP
      Host Port: 0/TCP
      Environment: <none>
      Mounts: <none>
      Volumes: <none>
  Conditions:
    Type Status Reason
    ----
    Available True MinimumReplicasAvailable
    Progressing True NewReplicaSetAvailable
    OldReplicaSets: <none>
    NewReplicaSet: nginx-deployment-7f555c9b4b (3/3 replicas created)
  Events:
    Type Reason Age From Message
    ----
    Available True MinimumReplicasAvailable
    Progressing True NewReplicaSetAvailable
    OldReplicaSets: <none>
    NewReplicaSet: nginx-deployment-7f555c9b4b (3/3 replicas created)
  Events:
    Type Reason Age From Message
    ----
```

Le libellé de l'application est "nginx" et a 3 répliques.

Le "nginx-deployment" aura des conteneurs nommés "nginx", basés sur l'image du menu fixe "nginx: 1.14.0", et exposera le port HTTP 80 par défaut.

Maintenant, vérifiez les pods Kubernetes et vous verrez le pod 'nginx-deployment-xxx' :

```
kubectl get pods
```

Ce qui nous donne :

```
root@dockerk8smaster:~/nginx# kubectl get pods
NAME READY STATUS RESTARTS AGE
nginx-deployment-7f555c9b4b-8lvw6 1/1 Running 0 6m54s
nginx-deployment-7f555c9b4b-8x8x1 1/1 Running 0 6m54s
nginx-deployment-7f555c9b4b-g9vrg 1/1 Running 0 6m54s
```

Vérifiez les détails du pod :

```
kubectl describe pods nginx-deployment-7f555c9b4b-8lvw6
```



Vous obtiendrez des « pods » de déploiement nginx avec 3 répliques sur les noeuds de travail :

```
root@dockerk8smaster:~/nginx# kubectl describe pods nginx-deployment-7f555c9b4b-8lvw6
Name: nginx-deployment-7f555c9b4b-8lvw6
Namespace: default
Priority: 0
Node: alexdocker01/192.168.8.100
Start Time: Fri, 22 Nov 2019 13:20:48 +0000
Labels: app=nginx
pod-template-hash=7f555c9b4b
Annotations: <none>
Status: Running
IP: 10.244.1.64
IPs:
IP: 10.244.1.64
Controlled By: ReplicaSet/nginx-deployment-7f555c9b4b
Containers:
nginx:
Container ID:
docker://42d09e0b09bf860b98c357be2cc2dc8e187347b0d4f16e3146df857ef7339063
Image: nginx:1.14.0
Image ID: docker-
pullable://nginx@sha256:8b600a4d029481cc5b459f1380b30ff6cb98e27544fc02370de836e397e34030
Port: 80/TCP
Host Port: 0/TCP
State: Running
Started: Fri, 22 Nov 2019 13:21:07 +0000
Ready: True
Restart Count: 0
Environment: <none>
Mounts:
/var/run/secrets/kubernetes.io/serviceaccount from default-token-2f46k (ro)
Conditions:
Type Status
Initialized True
Ready True
ContainersReady True
PodScheduled True
Volumes:
default-token-2f46k:
Type: Secret (a volume populated by a Secret)
SecretName: default-token-2f46k
Optional: false
QoS Class: BestEffort
Node-Selectors: <none>
Tolerations: node.kubernetes.io/not-ready:NoExecute for 300s
node.kubernetes.io/unreachable:NoExecute for 300s
Events:
Type Reason Age From Message
Normal Scheduled <unknown> default-scheduler Successfully assigned default/nginx-deployment-7f555c9b4b-8lvw6 to alexdocker01
Normal Pulling 8m1s kubelet, alexdocker01 Pulling image "nginx:1.14.0"
Normal Pulled 7m45s kubelet, alexdocker01 Successfully pulled image "nginx:1.14.0"
Normal Created 7m44s kubelet, alexdocker01 Created container nginx
Normal Started 7m44s kubelet, alexdocker01 Started container nginx
```



Ensuite, nous devons créer un nouveau service pour notre déploiement de Nginx.
Créez un nouveau fichier YAML nommé 'nginx-service.yaml'.

```
vim nginx-service.yaml
```

Collez la configuration ci-dessous.

```
apiVersion: v1
kind: Service
metadata:
  name: nginx-service
labels:
  run: nginx-service
spec:
  type: NodePort
  ports:
    - port: 80
  protocol: TCP
  selector:
    app: nginx
```

Sauvegarder et quitter.

EXPLICATIONS :

Nous créons un nouveau service kubernetes nommé 'nginx-service'.

Le type de service est 'NodePort' avec le port 80 par défaut de TargetPort HTTP.

Le service appartient à l'application nommée 'nginx' en fonction de notre déploiement 'nginx-deployment'.

Créez le service kubernetes à l'aide de la commande kubectl ci-dessous.

```
kubectl create -f nginx-service.yaml
```

Cela donne :

```
root@alexdockerk8smaster:~/nginx# kubectl create -f nginx-service.yaml
service/nginx-service created
```

Maintenant, vérifiez tous les services disponibles sur le cluster et vous obtiendrez le service Nginx dans la liste, puis vérifiez les détails du service.

```
kubectl get service
```



Cela donne :

```
root@dockerk8smaster:~/nginx# kubectl get service
NAME TYPE CLUSTER-IP EXTERNAL-IP PORT(S) AGE
kubernetes ClusterIP 10.96.0.1 <none> 443/TCP 143m
nginx-service NodePort 10.104.7.130 <none> 80:30291/TCP 97s
Puis pour plus d'informations :
kubectl describe service nginx-service
Ce qui nous donne :
root@alexdockerk8smaster:~/nginx# kubectl describe service nginx-service
Name: nginx-service
Namespace: default
Labels: run=nginx-service
Annotations: <none>
Selector: app=nginx
Type: NodePort
IP: 10.104.7.130
Port: <unset> 80/TCP
TargetPort: 80/TCP
NodePort: <unset> 30291/TCP
Endpoints: 10.244.1.64:80,10.244.1.65:80,10.244.2.3:80
Session Affinity: None
External Traffic Policy: Cluster
Events: <none>
Et vous verrez que le 'service nginx' NodePort est exécuté sur le port
'30291'.
Vérifiez en utilisant la commande curl sur tous les « workers ».
Sur le alexdocker01 et alexdocker02 :
Curl docker01:30291
```

Vous verrez la page par défaut de Nginx.

```
root@docker01:~# curl alexdocker01:30291
<!DOCTYPE html>
<html>
<head>
<title>Welcome to nginx!</title>
<style>
body {
width: 35em;
margin: 0 auto;
font-family: Tahoma, Verdana, Arial, sans-serif;
}
</style>
</head>
<body>
<h1>Welcome to nginx!</h1>
<p>If you see this page, the nginx web server is successfully installed and
working. Further configuration is required.</p>
<p>For online documentation and support please refer to
<a href="http://nginx.org/">nginx.org</a>.<br/>
Commercial support is available at
<a href="http://nginx.com/">nginx.com</a>.</p>
<p><em>Thank you for using nginx.</em></p>
</body>
</html>
```



L'installation et la configuration de Kubernetes Cluster sur Ubuntu 18.04 ont été effectuées avec succès.



4 Outils Docker

4.1 Orchestration et ordonnanceur

4.2 Intégration continue

4.3 Surveillance

4.4 Enregistrement

4.5 Sécurité

4.6 Stockage

4.7 Mise en réseau

4.8 Découverte de service

4.9 Build

4.10 Gestion graphique

