

FORMATION DOCKER

QU'ATTENDEZ-VOUS DE LA FORMATION ?

Qui êtes vous ?

Quel est votre profil ?

Où travaillez vous ?

Quel est votre parcours professionnel ?

Qu'attendez vous de la formation ?

DÉROULEMENT D'UNE JOURNÉE

09:30 - 11:00

Pause

11:15 - 12:45

Pause déjeuner

14:00 - 15:15

Pause

15:30 - 17:00

INTRODUCTION

PRÉSENTATION DE DOCKER

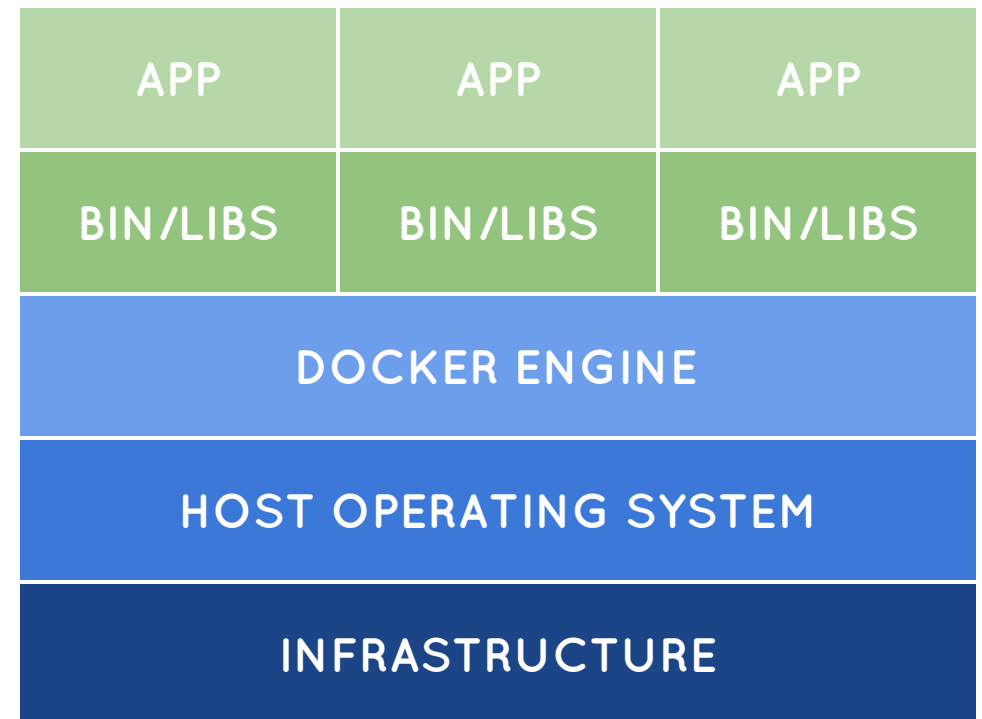
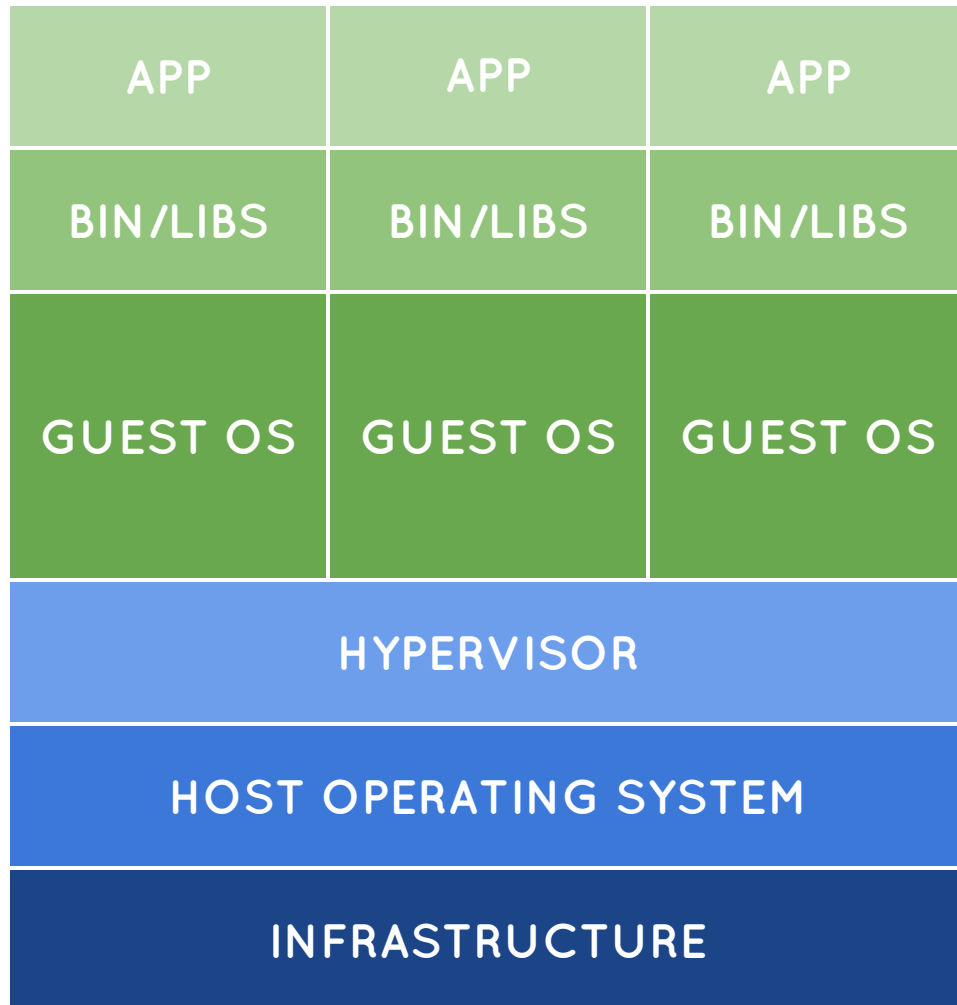
Utilise la technologie des conteneurs Linux

Distribue et exécute de manière isolée des applications

Exécutable depuis n'importe quelle machine Linux

Simple à prendre en main

CONTENEURS ET MACHINES VIRTUELLES



INSTALLATION DE DOCKER-CE

<https://store.docker.com/editions/community/docker-ce-desktop-mac>

<https://store.docker.com/editions/community/docker-ce-desktop-windows>

<https://docs.docker.com/engine/installation/linux/docker-ce/debian/>

<https://docs.docker.com/engine/installation/linux/docker-ce/ubuntu/>

L'ÉCOSYSTÈME DOCKER

D'autres outils sont fournis par Docker

Docker Hub / Docker Store : partager, gérer et construire des images en ligne

Docker Compose : définir des applications complexes

Docker Machine : déployer des applications sur plusieurs machines

Docker Cloud : service en ligne pour construire, tester, gérer et déployer des images sur ses serveurs

IMAGES ET CONTENEURS DOCKER

IMAGES ET CONTENEURS

Les images peuvent être construites, partagées et réutilisées

Les conteneurs sont construits à partir d'images

Plusieurs conteneurs peuvent partir d'une même image

Les conteneurs partagent les ressources avec la machine hôte

Les conteneurs sont isolés et embarquent leurs dépendances

RECHERCHE D'UNE IMAGE

Anciennement docker hub, docker store est le dépôt officiel d'images

On peut ainsi trouver et utiliser des images déjà prêtes à l'emploi

On peut aussi partir d'image existante pour en créer d'autres

On y trouve des images officielles, garantissant une image optimisée et à jour

The Docker Store

Find Trusted and Enterprise Ready Containers, Plugins, and Docker Editions

 Docker EE

 Docker CE

 Containers

 Plugins

Filters

TYPE

- ☒ Store
- ☐ Community (Docker Hub)

DOCKER CERTIFIED

- ☐  Docker Certified

CATEGORIES

- ☐ Analytics
- ☐ Application Frameworks
- ☐ Application Infrastructure
- ☐ Application Services
- ☐ Base Images

1 - 2 of 2 results for **nginx**. [Clear search](#)

Most Popular ▼



nginx

Docker | 10M+ Pulls



Official build of Nginx.
[Application Infrastructure](#)



kong

Docker | 1M+ Pulls



Open-source Microservice & API Management layer built on top of NGINX.
[Application Frameworks](#)

Not finding what you're looking for?

[View results from Community](#)

MANIPULATION D'IMAGES ET DE CONTENEURS

Usage: `docker COMMAND`

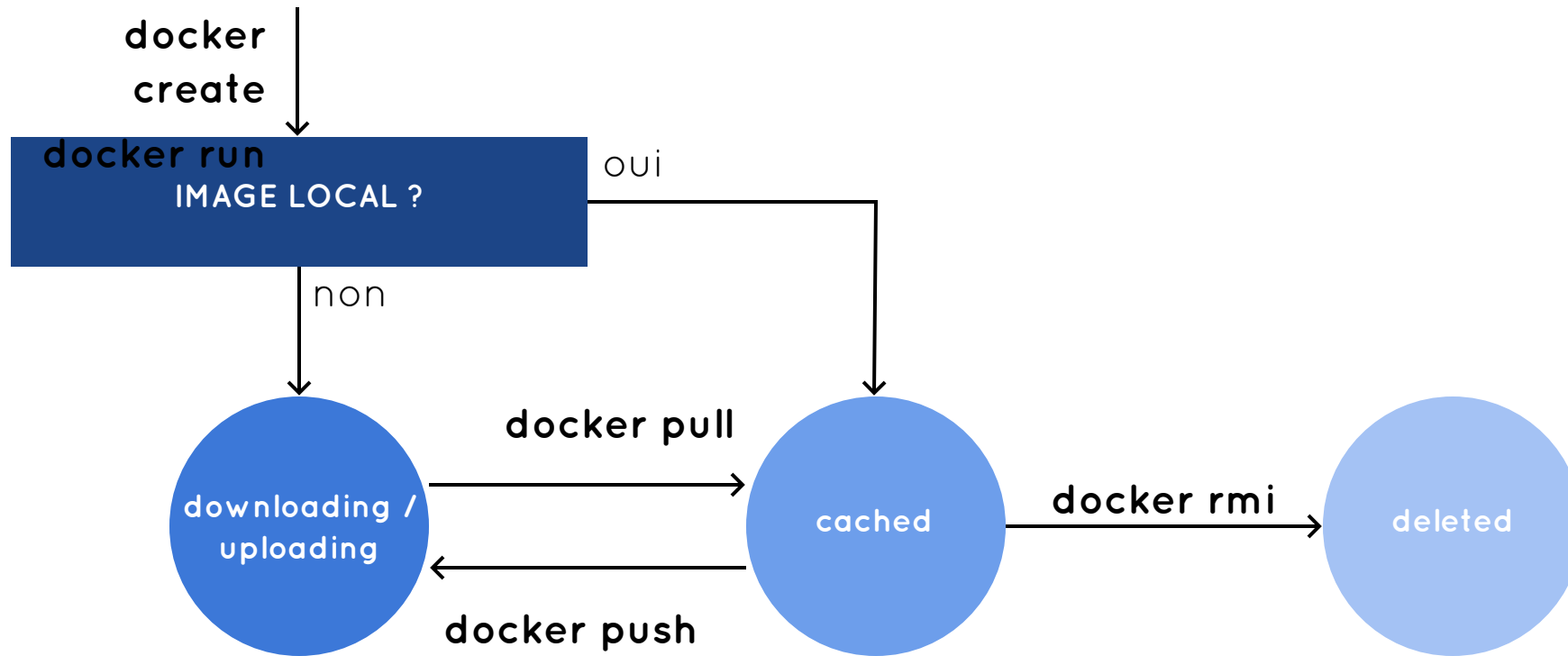
A self-sufficient runtime `for` containers

...

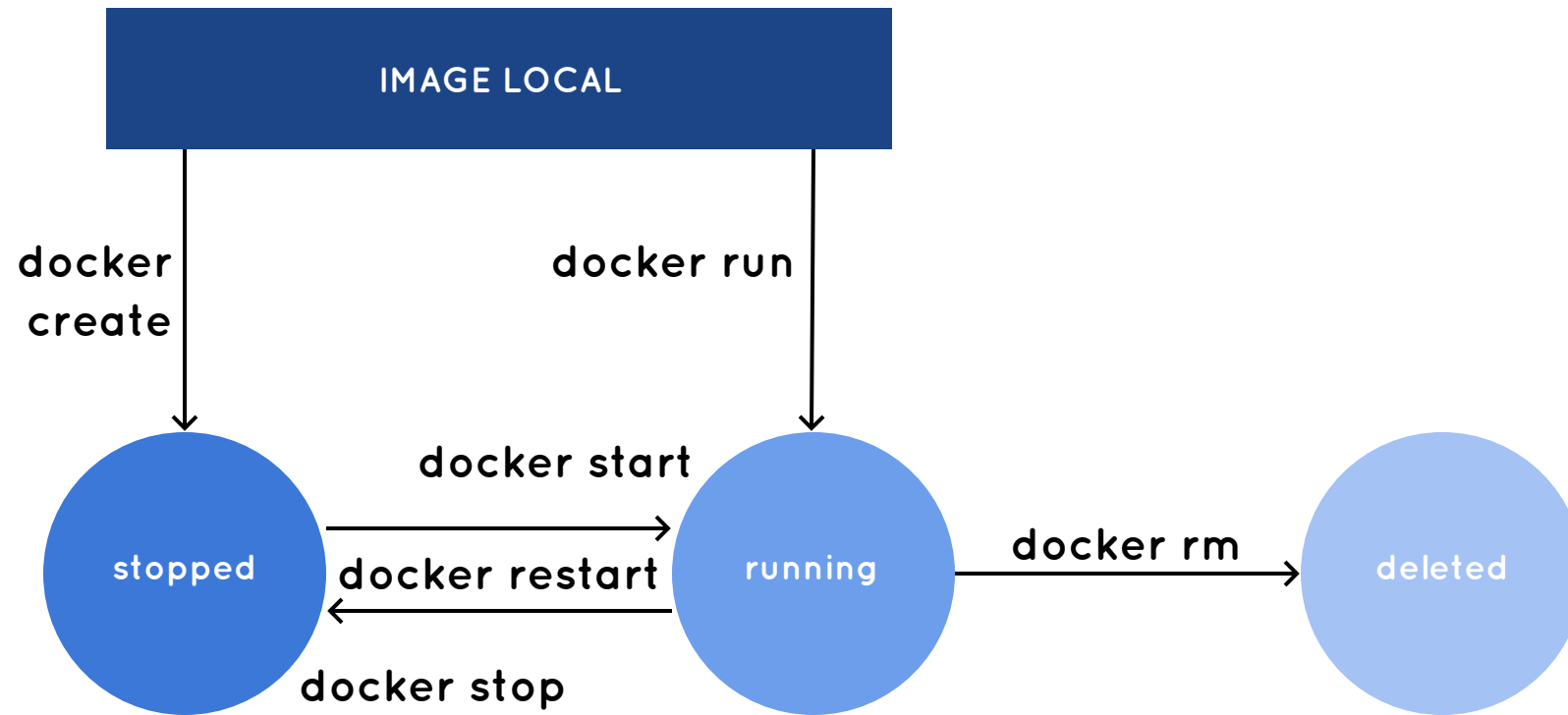
<code>create</code>	Create a new container
<code>exec</code>	Run a command <code>in</code> a running container
<code>pause</code>	Pause all processes within one or more containers
<code>ps</code>	List containers
<code>pull</code>	Pull an image or a repository from a registry
<code>push</code>	Push an image or a repository to a registry
<code>restart</code>	Restart one or more containers
<code>rm</code>	Remove one or more containers
<code>rmi</code>	Remove one or more images
<code>run</code>	Run a command <code>in</code> a new container
<code>start</code>	Start one or more stopped containers
<code>stop</code>	Stop one or more running containers
<code>unpause</code>	Unpause all processes within one or more containers
<code>version</code>	Show the Docker version information

Run `'docker COMMAND --help'` `for` more information on a command.

IMAGES DOCKER



CONTENEURS DOCKER



CRÉER SES PROPRES IMAGES : DOCKERFILE

Utiliser des images existantes n'est souvent pas suffisant

Docker offre la possibilité de créer ses propres images

Plutôt que de repartir de zéro, on peut partir d'une image déjà existante

On utilise un fichier nommé Dockerfile pour créer ses images

DOCKERFILE

```
FROM nginx:latest

MAINTAINER "Vincent CHAPRON"

RUN apt-get update \
    && apt-get install -y \
        wget \
        vim \
    && mkdir -p /workspace/docker

ADD 000-default.conf /etc/nginx/conf.d
```

```
$> docker build -t openska-docker .

# Sending build context to Docker daemon  2.048kB
# Step 1/4 : FROM nginx:latest
# [...]
# ---> b8efb18f159b
# Step 2/4 : MAINTAINER "Vincent CHAPRON"
# ---> Running in f250ac812653
# ---> a73d4523a1c1
# Removing intermediate container f250ac812653
# Step 3/4 : RUN apt-get update      && [...]
# ---> Running in 6342c4242ec1
# Get:1 http://security.debian.org stretch/[...]
# done.
# ---> c550f7362c79
# Removing intermediate container 6342c4242ec1
# Step 4/4 : ADD 000-default.conf /etc/nginx/conf.d
# ---> 4856bd0e69ff
# Removing intermediate container 5730afa68eee
# Successfully built 4856bd0e69ff
# Successfully tagged openska-docker:latest
```

PARTAGER SES IMAGES

```
$ # Create a docker ID https://cloud.docker.com/  
$ docker login  
  
$ # Build a Dockerfile, and the generated image  
$ docker build -t openska/php:7.1.9 .  
  
$ # Show all images  
$ docker images  
  
$ # Tag existing image  
$ docker tag openska/php:7.1.9 openska/php:latest  
  
$ # Publish image  
$ docker push openska/php:7.1.9  
$ docker push openska/php:latest
```

MANIPULATION DES CONTENEURS DOCKER

LIER DES CONTENEURS À NOTRE MACHINE

Un conteneur est par défaut isolé

Il est parfois nécessaire de lier nos conteneurs à notre machine

Comme par exemple, partager des volumes, lier des ports, ...

LIER DES CONTENEURS À NOTRE MACHINE

```
# CREATE or RUN container
$> docker create \
    -e ENV_VAR='Openska' \
    -v path/from/host:/path/to/container \
    -v other/path/from/host:/path/to/read-only:ro \
    -p 8080:80 \
    -p 8443:443 \
    -ti --rm --name my-container nginx:latest
```

Variable d'environnement
Partager des volumes
Partager des volumes en lecture seule
Lier des ports

DÉFINIR UN RÉPERTOIRE DE TRAVAIL

il est parfois nécessaire, ou tout simplement utile de définir un répertoire de travail à nos conteneurs

```
$> docker run -d -w /var/www --name nginx-server nginx:latest
# 66d4339a69fb7870578ff8161660d0023d1c86742b68564560ea8fbb7d329f59
$> docker exec -t nginx-server pwd
# /var/www
$> docker stop nginx-server
# nginx-server
$> docker rm nginx-server
# nginx-server
```

EXÉCUTER UNE COMMANDE

On peut vouloir exécuter une commande spécifique au lancement du conteneur

```
$> docker run -t --rm -w /etc/nginx/conf.d --name nginx-server nginx:latest cat default.conf
```

RETOUR SUR LE DOCKERFILE

Nous avons déjà vu deux mots-clés importants, **ADD** et **RUN**

On peut utiliser le Dockerfile pour préparer certaines actions sur le conteneur grâce à de nouveaux mots-clés, comme **ENV**, **EXPOSE**, **VOLUME** ou encore **WORKDIR**

RETOUR SUR LE DOCKERFILE

```
FROM nginx:latest

MAINTAINER "Vincent CHAPRON"

ENV ENV_VAR='Openska' FORMATION='Docker'

ENV VARIABLE 'value'

RUN apt-get update \
    && apt-get install -y \
        wget \
        vim \
    && mkdir -p /workspace/docker

ADD default.conf /etc/nginx/conf.d

EXPOSE 80 443 8000

VOLUME /var/www/new-volume

WORKDIR /var/www

CMD [ "ls", "-RA" ]
```

```
$> docker build -t openska-docker .

# docker build -t openska-docker .
# Sending build context to Docker daemon   2.56kB
#
# [...]
#
# Successfully built b55beeb15f4a
# Successfully tagged openska-docker:latest

$> docker run -t --rm openska-docker

# .:
# new-volume
#
# ./new-volume:
```

RÉSEAUX ET VOLUMES

On peut aussi vouloir connecter des conteneurs entre eux

Par exemple les conteneurs **PHP-FPM**, **NGINX** et **MySQL**

On peut alors partager des volumes pour, par exemple, utiliser les sockets UNIX

On peut aussi avoir un sous-réseau entre ces trois conteneurs pour utiliser des sockets réseaux ou des connexions TCP

RÉSEAUX ET VOLUMES

```
$ # DOCKER VOLUME
$ docker volume create my-volume
$ docker volume ls
$ docker run --rm -ti --name v1 -v my-volume:/workspace debian:stretch bash
$ docker run --rm -ti --name v2 -v my-volume:/data debian:stretch bash
$ docker container inspect v1

$ # DOCKER NETWORK
$ docker network create --subnet 132.76.0.0/24 my-network
$ docker network ls
$ docker run --rm -ti --network my-network debian:stretch bash
$     ip addr # In the container
$ docker network inspect my-network
```

CRÉATION DE STACKS AVEC DOCKER-COMPOSE

AUTOMATISER LA CRÉATION DE CONTENEURS

Partager une image, c'est simple, mais qu'en est-il de la manière dont on crée les conteneurs ?

Il existe plusieurs manières de créer un conteneur, quels volumes doivent être partagés, quels ports doivent être liés ?

Pour automatiser la création de conteneurs, on utilisera alors **Docker Compose**

PRÉSENTATION ET UTILISATION DE DOCKER-COMPOSE

Docker Compose fait partie de l'écosystème Docker et se compose d'une ligne de commande **docker-compose** et d'un fichier de configuration **docker-compose.yml** (nom par défaut)

Le fichier docker-compose.yml est un fichier de configuration qui permet entre autre de définir la création d'un conteneur

PRÉSENTATION ET UTILISATION DE DOCKER-COMPOSE

```
version: '3'
services:
  nginx:
    container_name: openska-nginx
    image: nginx:latest
    ports:
      - 8080:80
      - 8443:443
```

```
$> docker-compose up -d
# Creating network "docker_default" with the default driver
# Creating openska-nginx ...
# Creating openska-nginx ... done

$> docker-compose ps
#      Name                                Command                                State      [...]
# -----
# openska-nginx    nginx -g daemon off;                Up          [...]
```

```
$> docker-compose stop
# Stopping openska-nginx ... done

$> docker-compose rm -f
# Going to remove openska-nginx
# Removing openska-nginx ... done
```

On peut aussi utiliser des variables d'environnement, ainsi qu'un fichier .env dans notre docker-compose.yml

ENSEMBLE DE CONTENEURS ET DÉPENDANCES

On peut également créer une stack, c'est à dire **un ensemble fonctionnel de conteneurs**, qui fonctionnent ensemble avec des possibles dépendances entre eux

On pourra également utiliser docker-compose pour générer des réseaux et des volumes pour la stack entière

Tous les conteneurs d'une stack sont déjà dans un même sous réseau, et peuvent communiquer entre eux grâce à **leur nom de service**

DÉPLOYER DES CONTENEURS

GÉRER PLUSIEURS MACHINES AVEC DOCKER-MACHINE

Depuis notre machine, nous pouvons gérer plusieurs autres machines, virtuelles ou physiques, locales ou distantes grâce à **Docker Machine**

Plusieurs services proposent des facilités d'utilisation de Docker Machine comme Amazon Web Services ou Digital Ocean

Docker Machine s'utilise via la ligne de commande **docker-machine**

<https://docs.docker.com/machine/install-machine/#installing-machine-directly>

DIGITAL OCEAN

Cloud computing, designed for developers

Déploiement simple et rapide

Facture à l'heure et non au mois

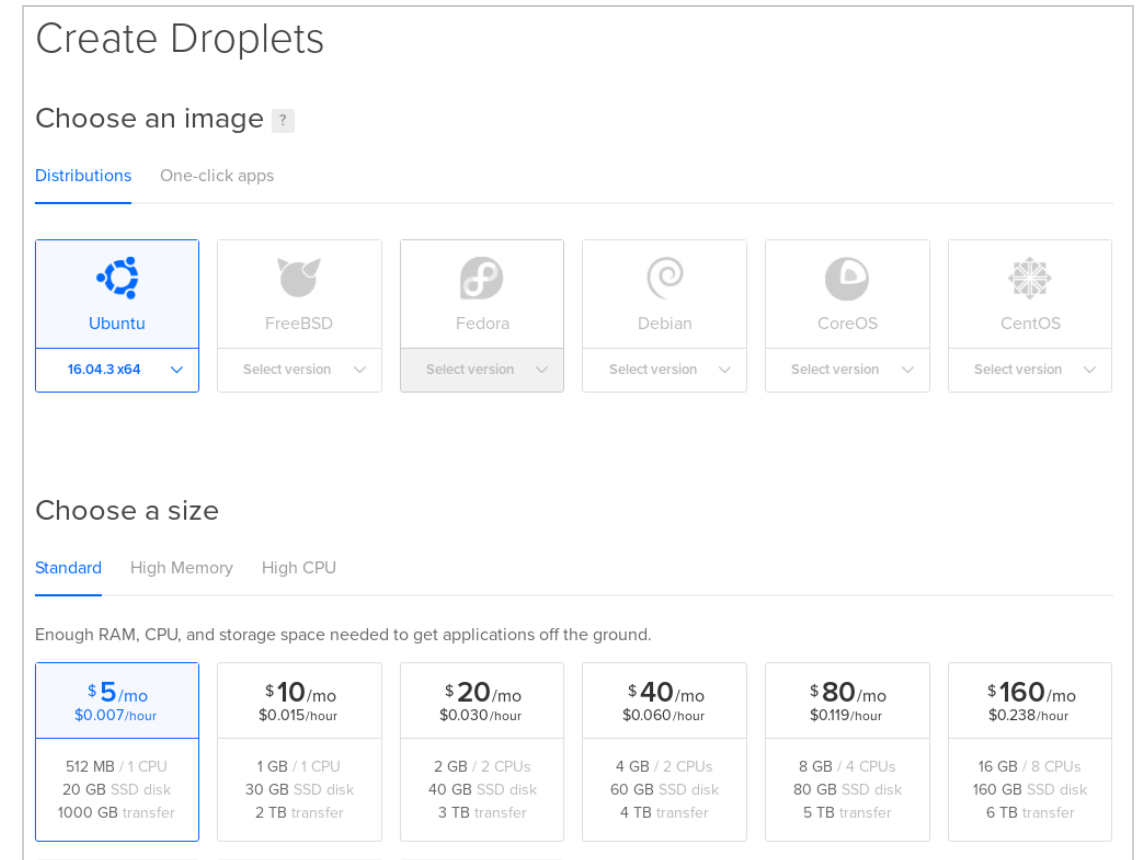
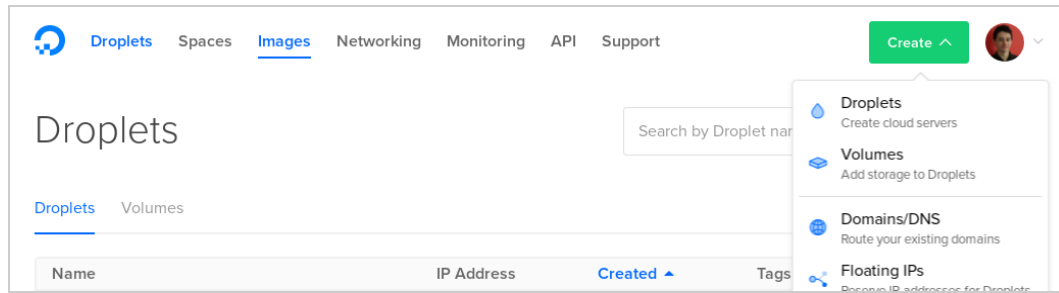
Une API pour accéder aux services à distances ...

<https://m.do.co/c/5c63165826a1>

Free credit active: You have a \$10 credit from a friend. Complete registration to get started.



DIGITAL OCEAN - CRÉATION D'UN DROPET



DIGITAL OCEAN - PRÉPARATION DU DROPLET

```
$ # On your droplet
$ groupadd --gid 1000 docker-user
$ useradd --uid 1000 --gid docker-user --shell /bin/bash --create-home docker-user
$ echo 'docker-user ALL=(ALL) NOPASSWD:ALL' >> /etc/sudoers

$ # CHANGE "PermitRootLogin yes" TO "PermitRootLogin no"
$ sed -i 's/\(PermitRootLogin\) yes/\1 no/' /etc/ssh/sshd_config

$ /etc/init.d/ssh restart

$ # Add your SSH key
mkdir -p /home/docker-user/.ssh
echo "CHANGE WITH PUBLIC KEY" >> /home/dockert-user/.ssh/authorized_keys
```

DÉPLOYER DES CONTENEURS SUR SES MACHINES

On ajoute les machines distantes grace à la ligne de commande docker-machine

Dokcer Machine installe Docker sur la machine distante si celui ci n'est pas installé

On peut directement déployer des conteneurs depuis notre machine vers la machine distante

DÉPLOYER DES CONTENEURS SUR SES MACHINES

```
$ # On your machine
$ docker-machine create \
  --driver generic \
  --generic-ip-address 46.101.240.47 \
  --generic-ssh-key ~/.ssh/id_rsa \
  --generic-ssh-user docker-user \
  openska-formation

$ # On your droplet
sudo usermod -a -G docker docker-user

$ # On your machine
$ eval $(docker-machine env openska-formation)
$ docker run --rm -ti -p 80:80 nginx:latest
# Go to http://46.101.240.47/
```

LOAD-BALANCING : SWARM MODE

On peut déployer nos différents conteneurs sur différentes machines, et éventuellement les répliquer

De plus, depuis la version 3 des fichiers docker-compose, on peut directement donner les instructions de déploiement à nos services dans notre fichier docker-compose.yml

<https://docs.docker.com/compose/compose-file/#deploy>

SWARM MODE : AJOUTER UN MANAGER ET DES NODES

```
$ # On définit une machine comme manager
$ docker swarm init --advertise-addr 46.101.38.171

$ # On définit des machines comme nodes
$ docker swarm join --token \
    SWMTKN-1-6ct3tlj8pfeb11fdunmts84ci297kle4k04m4hfuqnrheukyu-d4gyc1cs8umuixmaivlboa4c \
    46.101.38.171:2377
```

SWARM MODE : DOCKER COMPOSE

```
version: '3'
services:
  app:
    image: nginx:latest
    ports:
      - 80:80
      - 443:443
    deploy:
      mode: replicated
      replicas: 4
```

```
# On deploy notre stack
$ docker stack deploy \
  --compose-file docker-compose.yml \
  my-stack

# On surveille notre stack
$ docker stack ps my-stack
$ docker stack services my-stack

# On détruit notre stack
$ docker stack rm my-stack
```

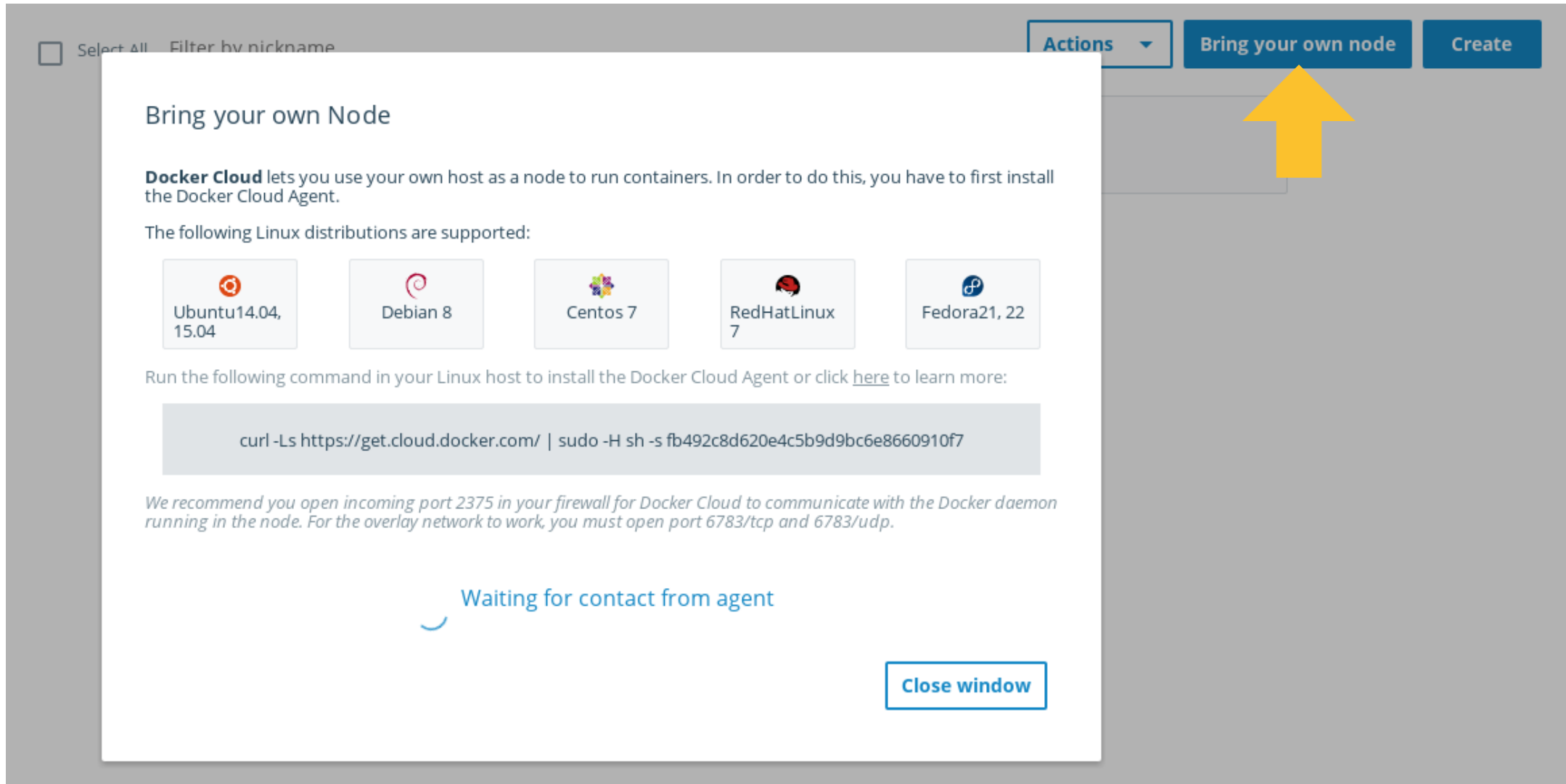
DOCKER CLOUD

PRÉSENTATION DE L'OUTIL

Docker Cloud est un ensemble de services et d'outils pour gérer des applications Docker en intégration continue

On peut gratuitement avoir des images docker public, une seule image docker privée, un seul serveur de déploiement

AJOUTER UNE NODE



☐ Select All Filter by nickname Actions Bring your own node Create

Bring your own Node

Docker Cloud lets you use your own host as a node to run containers. In order to do this, you have to first install the Docker Cloud Agent.

The following Linux distributions are supported:

Ubuntu14.04,
15.04

Debian 8

Centos 7

RedHatLinux
7

Fedora21, 22

Run the following command in your Linux host to install the Docker Cloud Agent or click [here](#) to learn more:

```
curl -Ls https://get.cloud.docker.com/ | sudo -H sh -s fb492c8d620e4c5b9d9bc6e8660910f7
```

We recommend you open incoming port 2375 in your firewall for Docker Cloud to communicate with the Docker daemon running in the node. For the overlay network to work, you must open port 6783/tcp and 6783/udp.

Waiting for contact from agent

Close window

DÉPLOYER UN SERVICE

←

docker cloud

+ Get Help vincentchapron

Services / Wizard

BETA
Swarm Mode

BUILD

Repositories

APPLICATIONS

Stacks

Services

Containers

INFRASTRUCTURE

Node Clusters

Nodes

SETTINGS

Cloud Settings

Status

General settings

IMAGE nginx latest

SERVICE NAME nginx-56ad5555

NICKNAME The alias of your service

ADD TO STACK Select a Stack

CONTAINERS 1

DEPLOYMENT STRATEGY Emptiest Node

DEPLOYMENT CONSTRAINTS Select...

AUTORESTART ☒ Off ☐ On failure ☐ Always

AUTODESTROY ☒ Off ☐ On success ☐ Always

SEQUENTIAL DEPLOYMENT ☐ AUTOREDEPLOY ☐

NETWORK bridge

SUMMARY

General settings

Container configuration

Ports


Links

Environment variables

Volumes

Create & Deploy


CRÉER UN DÉPÔT

 docker cloud

+

Get Help

▼

 vincentchapron ▼

Repositories / Create


Create Repository


vincentchapron / Name

Description

Visibility


Using 0 of 1 private repositories. [Get more](#)


☒ **Public** 
Public repositories appear in Docker Store search results

☐ **Private** 
Only you can see private repositories

Build Settings *(optional)*

Autobuild triggers a new build with every **git push** to your source code repository [Learn more](#)

 Connected

 Disconnected

Cancel

Create

Create & Build

Pro tip

You can always push a new image to this repository using the CLI

```
$ docker tag local-image:tagname new-repo:tagname
$ docker push new-repo:tagname
```

Make sure to change tagname with your desired image repository tag.

LIER UN DEPOT À GITHUB

 × ▼



▼

Required

▼ [Click here to customize the build settings](#)

BUILD RULES +

The build rules below specify how to build your source into Docker images.

Source Type	Source	Docker Tag	Dockerfile location	Build Caching	
Branch ▼	master	latest	Dockerfile	<input checked="" type="checkbox"/>	
Tag ▼	/^v[0-9.]+\$/	{\1}	Dockerfile	<input checked="" type="checkbox"/>	

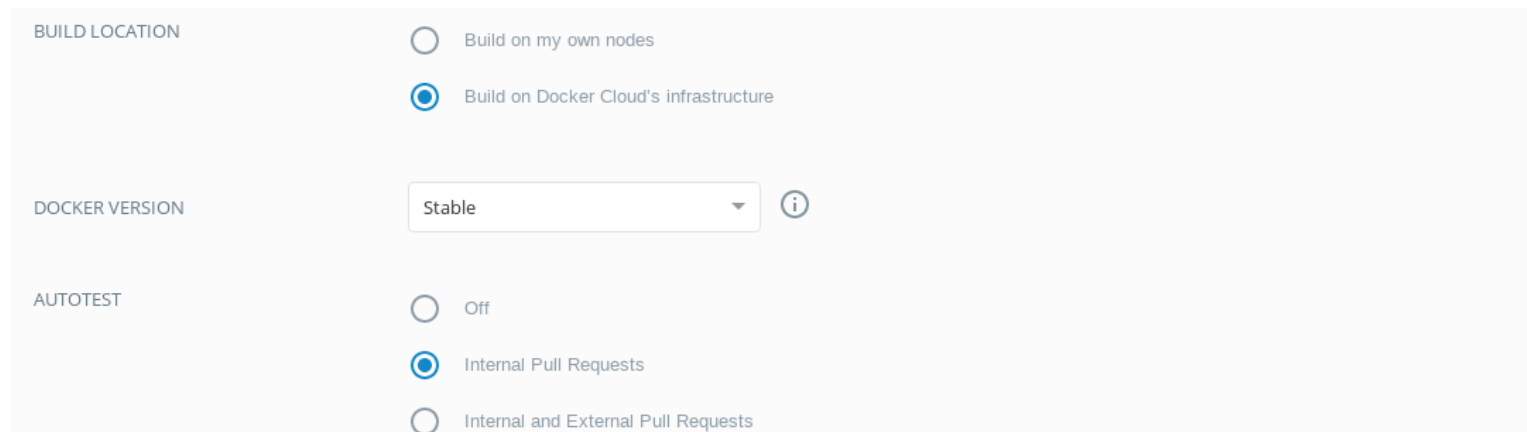
▼ [View example build rules](#)

Scenario	Source Type	Source	Docker Tag	Matches	Docker Tag Built
Exact match	Branch	master	latest	master	latest
Match versions	Tag	/^[0-9.]+\$/	release-{sourceref}	1.2.0	release-1.2.0
Trailing modifiers	Tag	/^[0-9.]+/	release-{sourceref}	1.2.0-rc	release-1.2.0-rc
Extract version number	Tag	/^v[0-9.]+\$/	version-{\1}	v1.2.3	version-1.2.3

DÉPLOIEMENT CONTINUE

Grace à nos dépôts et leurs liaisons à github, nous pouvons automatiquement déployer nos images et aussi lancer les tests automatiquement

<https://github.com/vincent-chapron/example-docker-tests>



The image shows a configuration interface for Docker Cloud builds. It has three main sections: BUILD LOCATION, DOCKER VERSION, and AUTOTEST. Each section has radio buttons or a dropdown menu to select options.

Section	Option	Selected
BUILD LOCATION	Build on my own nodes	No
	Build on Docker Cloud's infrastructure	Yes
DOCKER VERSION	Stable	Yes
AUTOTEST	Off	No
	Internal Pull Requests	Yes
	Internal and External Pull Requests	No

SÉCURITÉ ET LIMITES DES CONTENEURS

DE QUOI FAUT-IL AVOIR CONCSIENCE ?

Il est important que garder en tête certaines vulnérabilités concernant l'utilisation des conteneurs, en voici une liste non-exhaustive

- vulnérabilité du Kernel
- denial of service (DoS)
- containers breakouts (improbable, mais possible)
- images empoisonnées

PROTÉGER SES CONTENEURS

Voici quelques règles de sécurité pour protéger ses conteneurs en production

- les conteneurs devraient être sur une machine virtuelle ou sur un hôte dédié
- seul le loadbalancer/proxy doit exposer un port
- toutes les images doivent définir un utilisateur et ne doit pas fonctionner en tant que root
- la provenance de toutes les images doit être vérifié

PROTÉGER SES CONTENEURS

- ajouter un sytem de monitoring/alerting pour détecter le trafic inhabituel
- les conteneurs devraient fonctionner avec uniquement des logiciels à jour
- le server peut se voir ajouter un logiciel de sécurité comme SELinux ou AppArmor
- Les binaires des conteneurs devraient être supprimés, par exemple setuid ou setgid
- Le partage de volumes devrait se faire en read-only dès que possible

PROTÉGER SES CONTENEURS

- limiter les capacités du kernel dans les conteneurs,
<http://man7.org/linux/man-pages/man7/capabilities.7.html>
- limiter les performances des conteneurs
 - la mémoire maximum par conteneur
 - le temps d'usage du cpu par conteneur
 - le nombre de cpu par utilisateur
 - le nombre de file descriptor par conteneur
- limiter le nombre de redémarrage

AVEZ VOUS DES
QUESTIONS ?

MERCI