



FORMATION DOCKER SWARM



Présentation par

Daniel LAVOIE

Architecte / Talent Development Manager

+33 (0)6 45 97 73 33

daniel.lavoie@invivoo.com

Plan de Cours

Introduction aux conteneurs

Créer ses premiers conteneurs Docker

Les images Docker

Le réseau avec Docker

La persistance des données avec Docker

L'écosystème Docker

Concepts avancés

Introduction aux conteneurs

Présentation du concept de conteneur Linux

Cas d'utilisation des conteneurs Linux








Les différences entre conteneurs et machines virtuelles

Présentation de Docker et de son architecture

Un état des lieux de la livraison logicielle

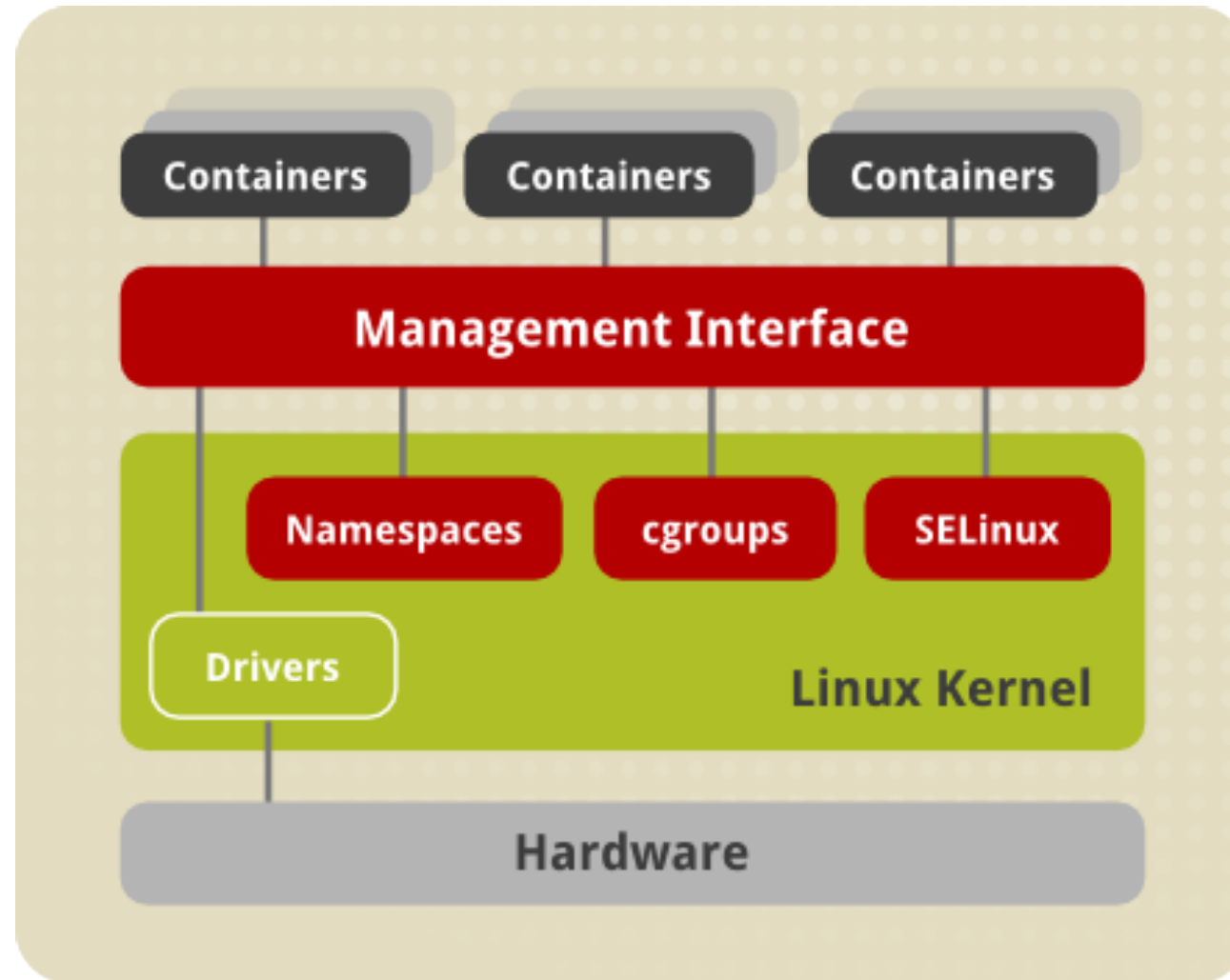
- Livrer du code... c'est un art !
- De plus en plus de dépendances externes
- La légende de l'environnement de dev ISO-Prod. Tout le monde en a entendu parler mais personne ne l'a jamais vu...
- The Matrix from HELL !

The Matrix from HELL !













Static website	?	?	?	?	?	?	?
Web frontend	?	?	?	?	?	?	?
Background workers	?	?	?	?	?	?	?
User DB	?	?	?	?	?	?	?
Analytics DB	?	?	?	?	?	?	?
Queue	?	?	?	?	?	?	?
							

Il était une fois... un conteneur

Un petit rappel sur les conteneurs Linux



Débutons par une analogie

	?	?	?	?	?	?
	?	?	?	?	?	?
	?	?	?	?	?	?
	?	?	?	?	?	?
	?	?	?	?	?	?
	?	?	?	?	?	?
						

source : <https://www.youtube.com/watch?v=Q5POuMHxW-0>

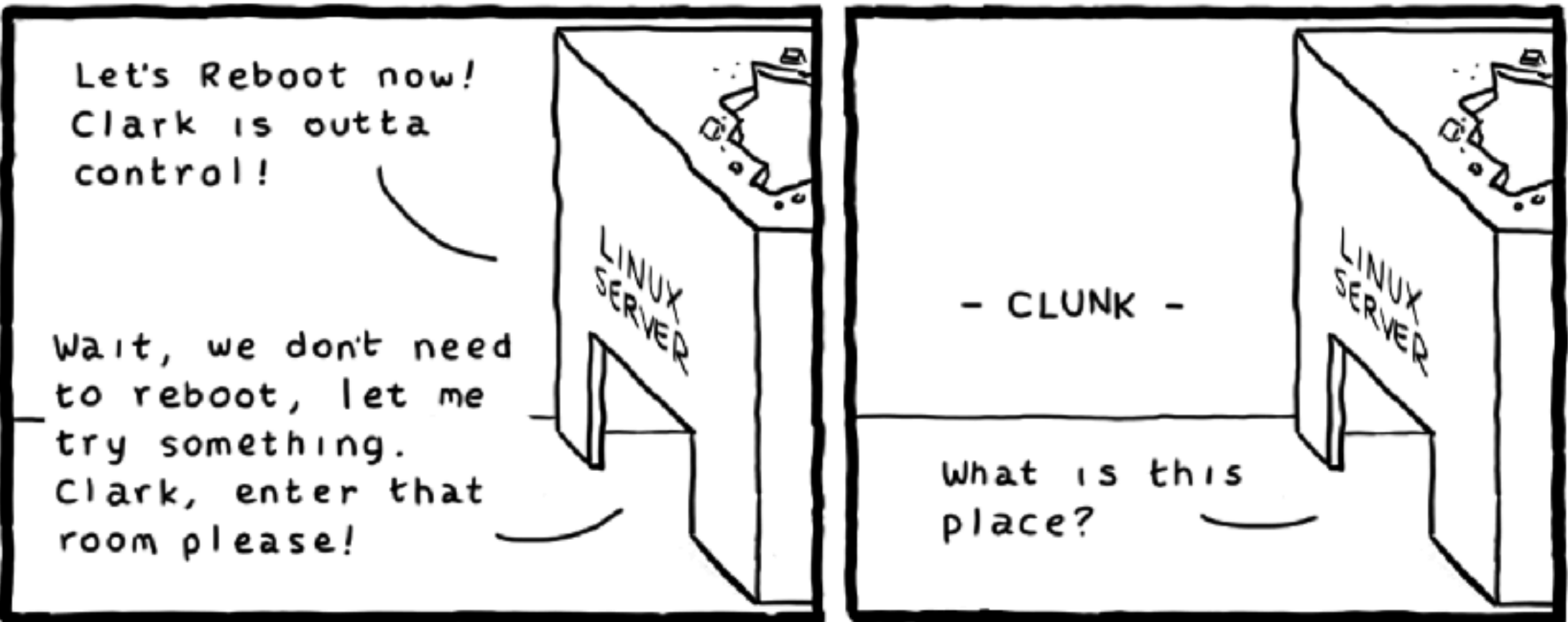
Débutons par une analogie



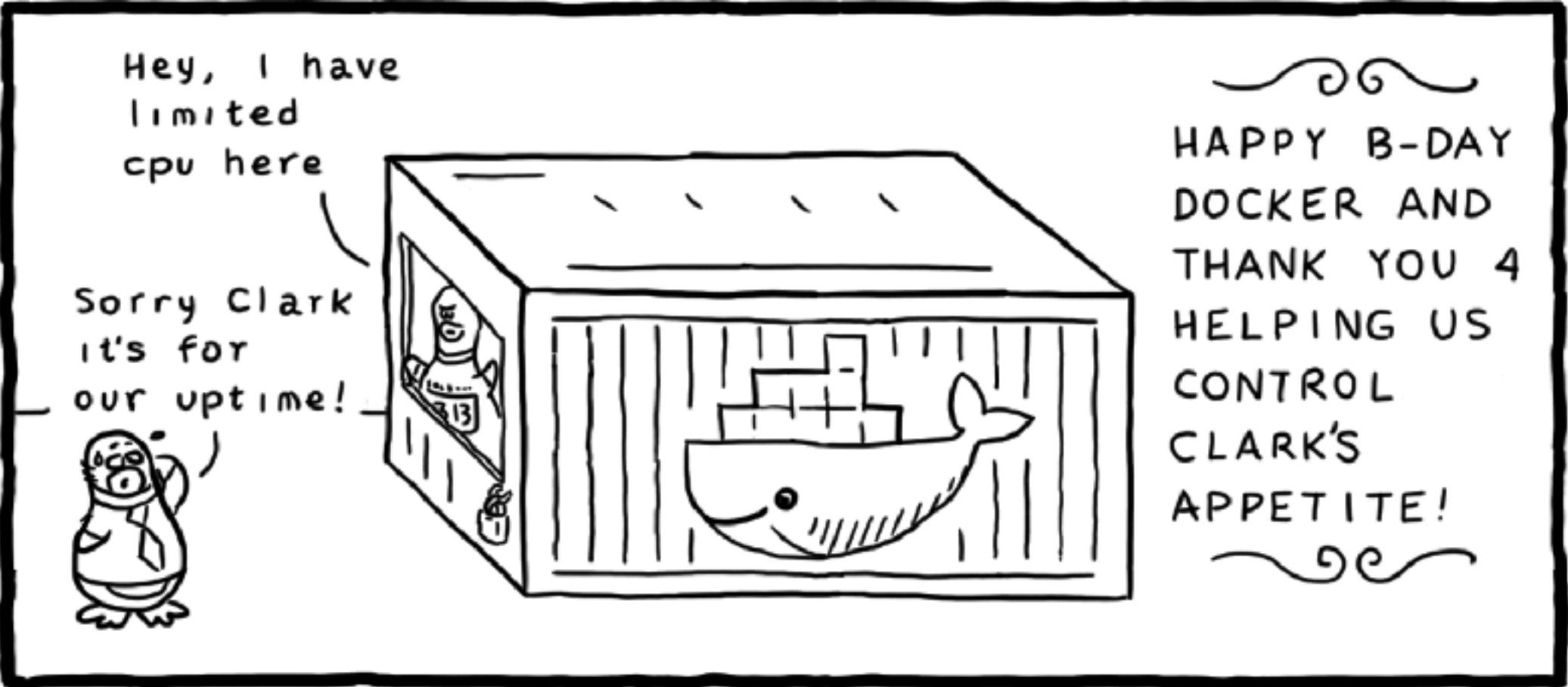
source : <https://www.youtube.com/watch?v=Q5POuMHxW-0>

Multi-assets
 FULL STACK
 Systems critiques
 FLOW BUSINESS
 ENABLER
 CLIENTS EXIGEANTS
 CHANGEMENT MONDIAL
 Cloud Grid
 Computing
 AGILITE
 Business Intelligence
 COMPOSER
 ENGAGEMENT

Clark is hungry



Clark is hungry



Daniel Stori {turnoff.us}

Multi-assets
 FULL STACK
 ENABLER
 Systems critiques
 FLOW BUSINESS
 AGILITE
 Cloud Grid
 Computing
 Business Intelligence
 COMPOS "BUT"
 ENGAGEMENTS
 ENTS EXIGEANTS
 FINANCIELLE MONCE

L'histoire de Docker

- Solomon Hykes fonde dotcloud en 2008 à Montrouge.
- dotcloud permet l'hébergement de conteneur Linux
- Fort de leur expérience de gestion de conteneur, création du projet Docker.



Quelques outils sur lequel Docker repose

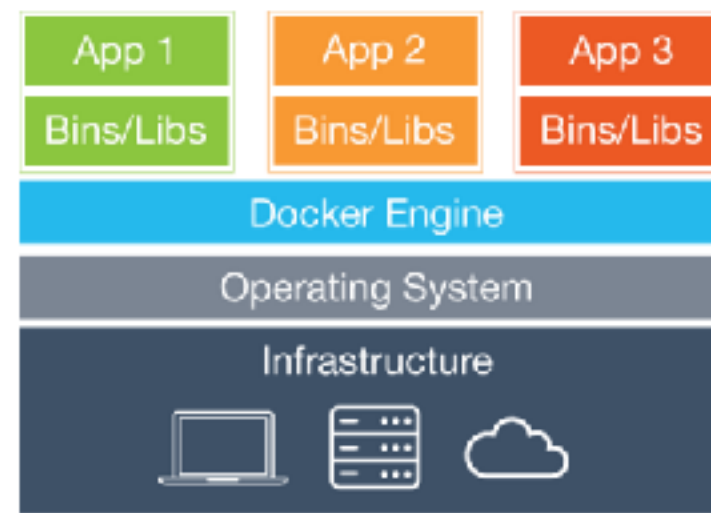
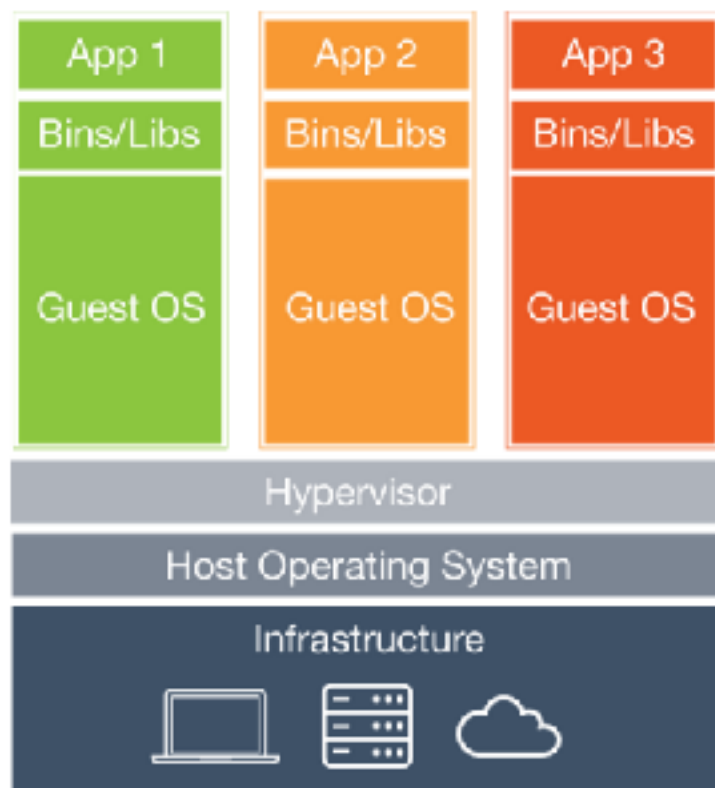
- Kernel Namespaces
- Cgroups
- IPTables
- Union File Systems
- libcontainer

Docker n'a rien inventé, par contre, il simplifie la gestion de conteneur Linux

... bref, Docker c'est ...



Et une simple VM ne serait pas plus simple ?

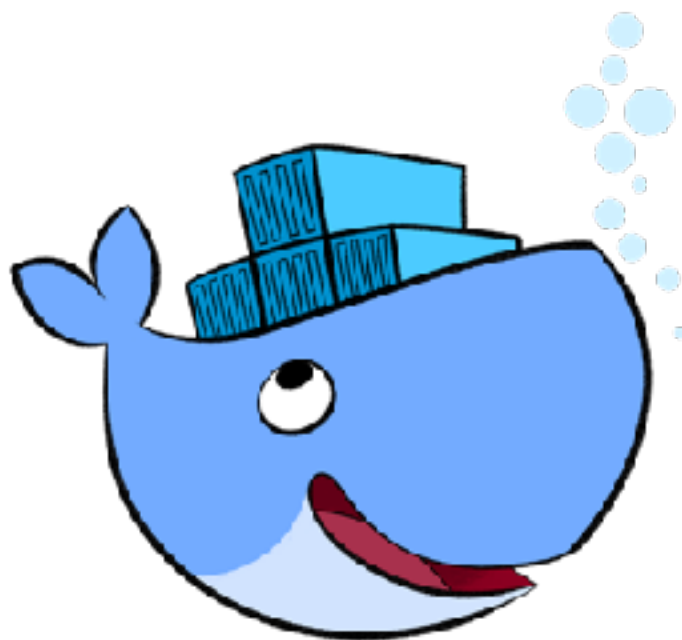


source : <https://www.docker.com>

Pourquoi Docker et pas une autre solution de conteneurs ?

- Image management
- Resource Isolation
- File System Isolation
- Network Isolation
- Change Management
- Sharing
- Process Management
- Service Discovery (DNS depuis 1.10)

Gestionnaire de conteneur - Docker Engine



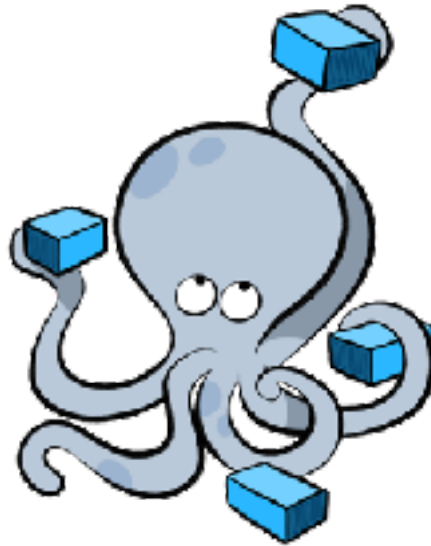
Toute une plate-forme

Docker Machine



Installateur Docker

Docker Compose



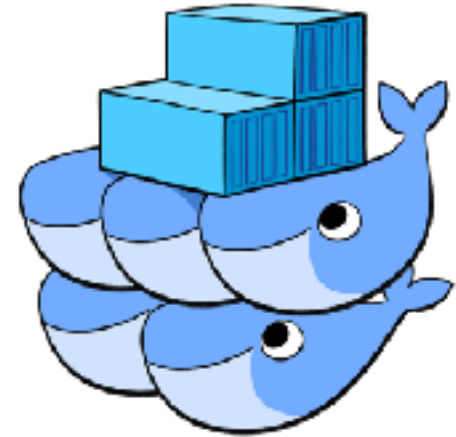
Gestionnaire
d'environnement

Docker Registry



Dépôt de
conteneur

Docker Swarm



Gestionnaire
de cluster

Vocabulaire Docker

- **Host**

- Représente un serveur sous lequel le Docker Engine est installé. Cette machine gère donc des conteneurs

- **Image**

- Représente une capture d'un Filesystem associé avec des paramètres. Une image est sans état et ne change jamais. Un conteneur représente l'instance d'une image.

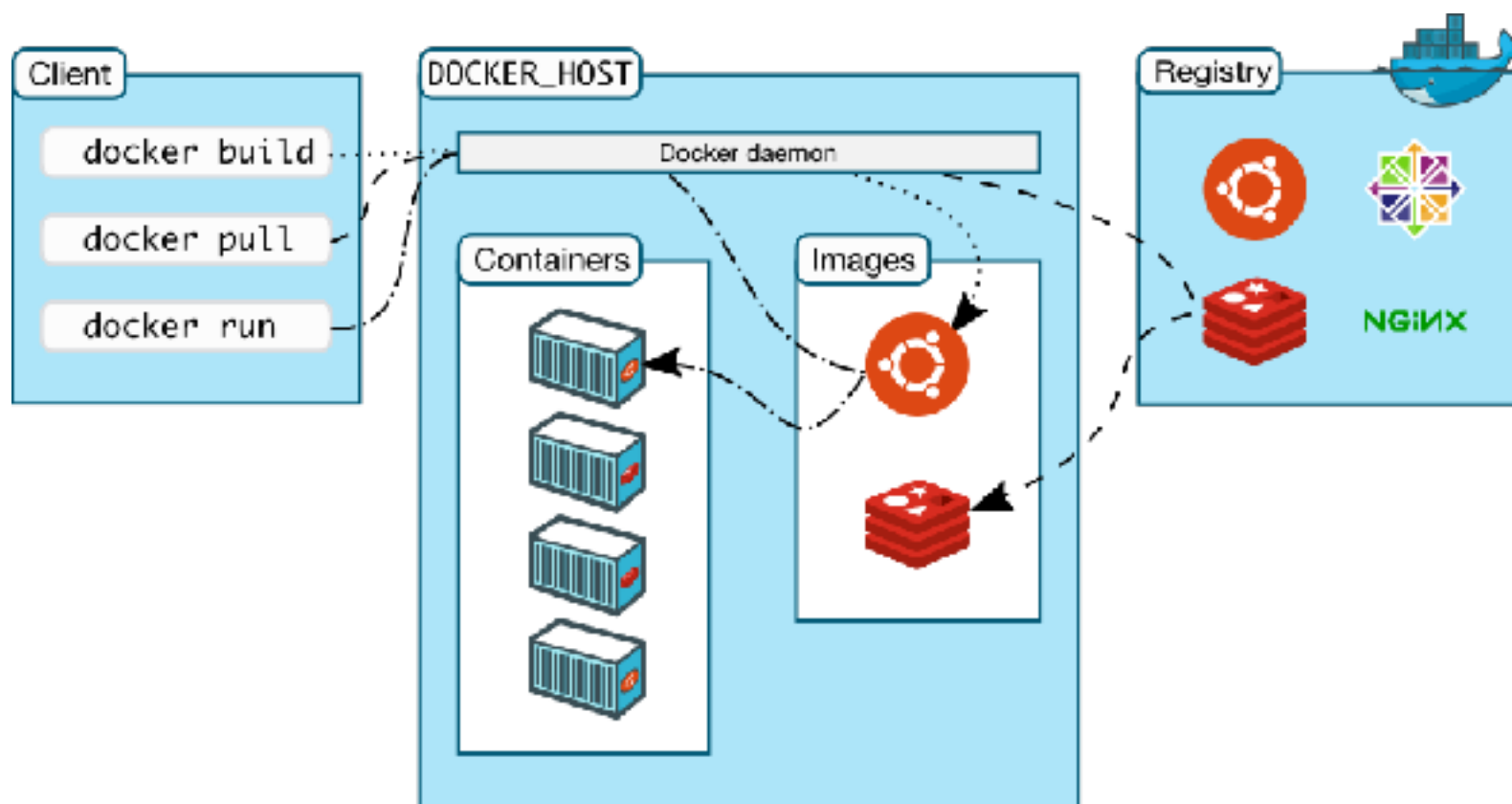
- **Volume**

- Filesystem pouvant être associé à un ou plusieurs conteneur. Le volume est géré par le Docker Engine. Il est par conséquent persisté alors que les conteneurs sont éphémère.

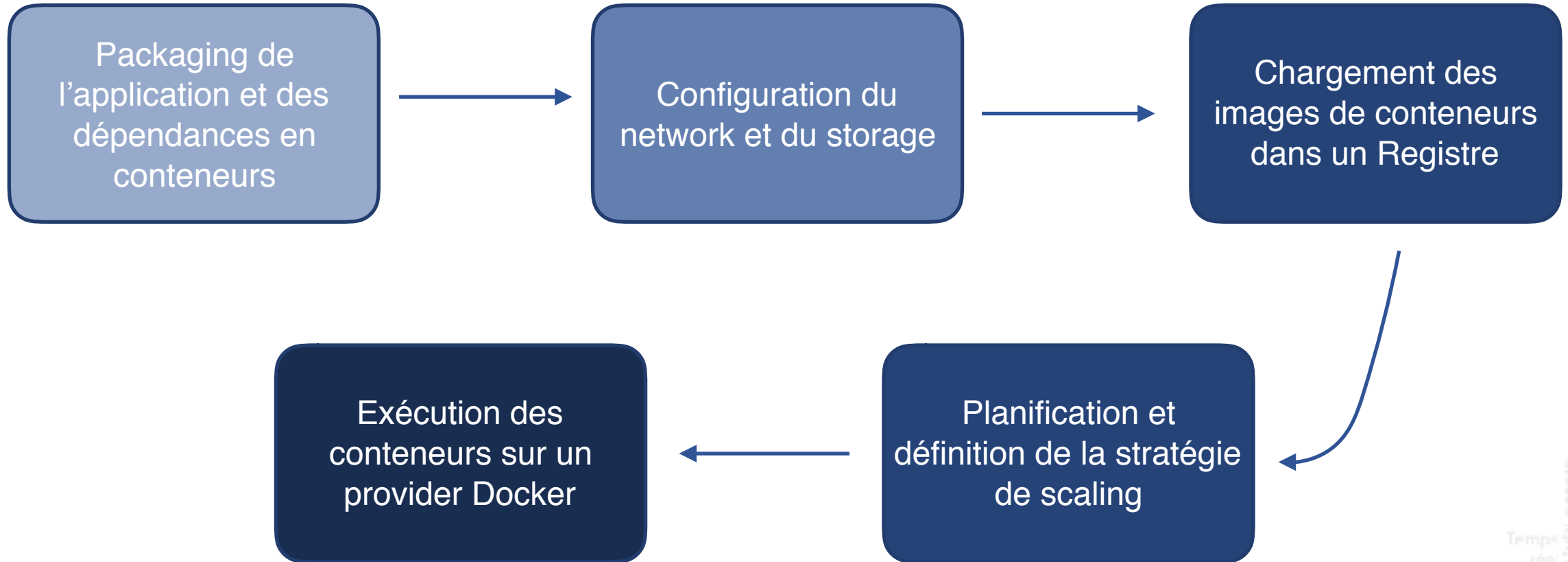
- **Network**

- Sous réseau virtuels créés par le Docker Engine pour permettre l'isolation réseautique des conteneurs sur une même machine.

Architecture du Docker Engine



Cinématique typique d'une mise en place de Docker



Créer des premiers conteneurs Docker

Installation de Docker

Le cycle de vie d'un conteneur

Lancer un conteneur avec docker run (en mode interactif, en mode détaché...)

Intéragir avec un conteneur depuis le host (exec, inspect, logs...)

Installation de Docker

<https://docs.docker.com/engine/installation/>

Lancement d'un conteneur Debian

```
root@docker-invivoo:~# docker run -it debian bash
Unable to find image 'debian:latest' locally
latest: Pulling from library/debian
```

```
5040bd298390: Already exists
Digest: sha256:abbe80c8c87b7e1f652fe5e99ff1799cdf9e0878c7009035afe1bccac129cad8
Status: Downloaded newer image for debian:latest
root@a425a78fae15:/#
```

Lancement d'un conteneur

Attachement
au terminal

Commande à lancer une fois
le conteneur démarré

Client Docker

Image du conteneur

```
root@docker-invivoo:~# docker run -it debian bash
```

Modes de lancements d'un conteneur

- En premier plan

```
root@docker-invivoo:~# docker run -it debian bash
```

L'option -it permet d'obtenir un terminal interactif sur l'exécution du conteneur.

Le conteneur sera stoppé lorsque le process lancé par le conteneur se terminera.

- Mode détaché

```
root@docker-invivoo:~# docker run -d debian bash
5b93ac8a44ac945a2bae918f2e740b39e89bc8fb3b0cb6281093d4a08790a519
root@docker-invivoo:~#
```

L'option -d permet de lancer le conteneur et de le laisser tourner en background.

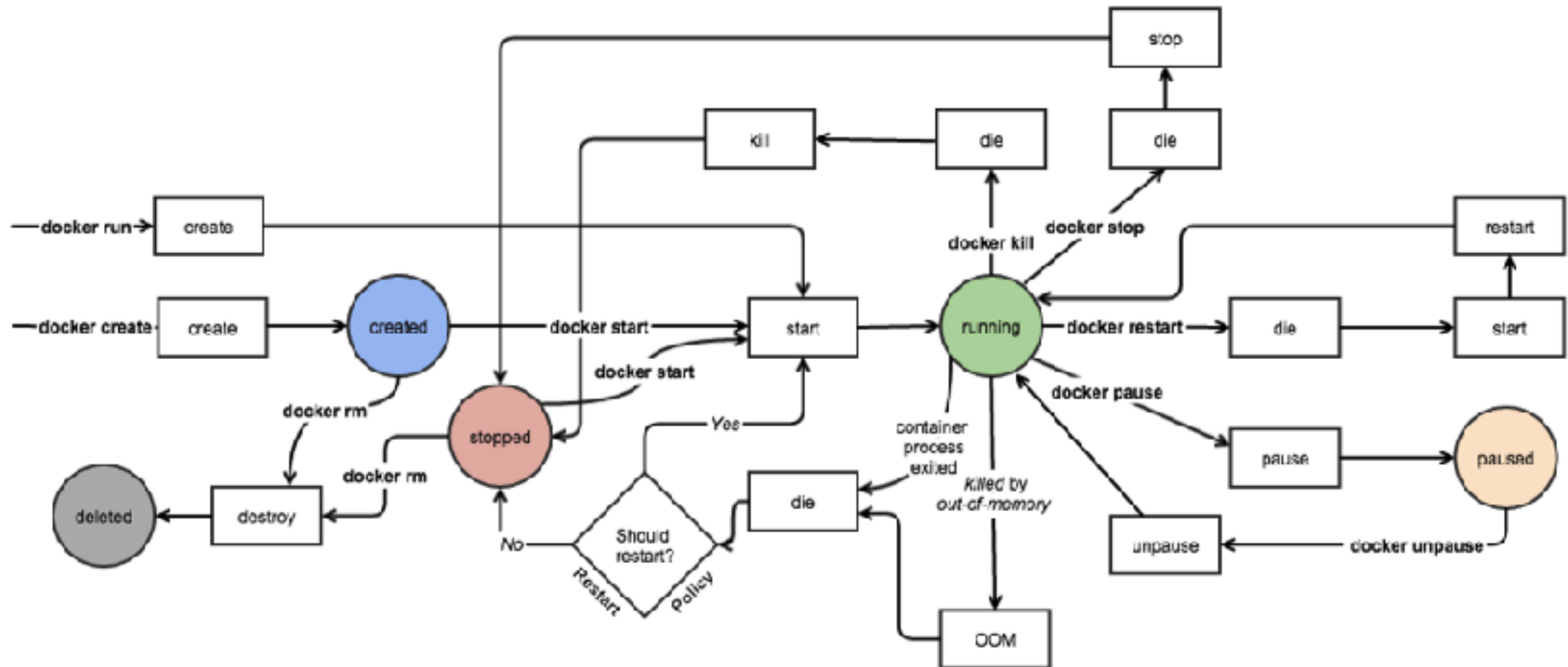
Comme le mode premier plan, le conteneur sera stoppé une fois sa commande complétée.

Exemple de lancement classique

```
root@docker-invivo:~# docker run -d nginx
Unable to find image 'nginx:latest' locally
latest: Pulling from library/nginx
5040bd298390: Already exists
d7a91cdb22f0: Pull complete
9cac4850e5df: Pull complete
Digest: sha256:33ff28a2763feccc1e1071a97960b7fef714d6e17e2d0ff573b74825d0049303
Status: Downloaded newer image for nginx:latest
474944b57327757b14c2bf1d4b6c4e6576c077d39665df7b7da56940821f2617
```

- Cet exemple démarre un conteneur avec un serveur Nginx.
- La documentation de l'image nginx nous indique que le conteneur tourne en arrière plan en écoutant les requêtes sur le port 80.

Cycle de vie d'un conteneur



Interactions avec les conteneurs - Cycle de vie

- Arrêt d'un conteneur

```
$ docker stop conteneur
```

- Démarrage d'un conteneur arrêté

```
$ docker start conteneur
```

- Redémarrage d'un conteneur

```
$ docker restart conteneur
```

- Suppression d'un conteneur

```
$ docker rm conteneur
```

Interactions avec les conteneurs - Cycle de vie

- Kill d'un conteneur

```
$ docker kill conteneur
```

- Pause d'un conteneur

```
$ docker pause conteneur
```

- Reprise d'un conteneur en pause

```
$ docker unpause conteneur
```


Inspection d'un conteneur

```
root@docker-invivoo:~# docker inspect 474944b57327
[
  {
    "Id": "474944b57327757b14c2bf1d4b6c4e6576c077d39665df7b7da56940821f2617",
    "Created": "2017-01-23T18:32:08.572673533Z",
    "Path": "nginx",
    "Args": [
      "-g",
      "daemon off;"
    ],
    "State": {
      ...
    },
    ...
    "Config": {
      "Hostname": "474944b57327",
      ...
    },
    "NetworkSettings": {
      ...
    }
  }
]
```

Inspection d'un conteneur

- La commande **docker exec** permet de lancer un processus sur un conteneur. Un cas d'utilisation récurrent consiste à lancer un terminal avec **bash** pour ensuite s'attacher au conteneur afin de pouvoir le débiter de l'intérieur.

```
root@docker-invivo:~# docker exec -it prickly_tesla bash
root@474944b57327:/#
```

**EVERY TIME YOU DOCKER EXEC -IT
BASH**



**GOD KILLS A
KITTEN**

memegenerator.net

Récupération des logs d'un conteneur

- Affichage en entier des logs du conteneur

```
$ docker logs conteneur
```

- Simulation d'un tail sur les logs du conteneur

```
$ docker logs -f conteneur
```

- Affichage des logs depuis un moment donnée

```
$ docker logs --since=3h conteneur
```

Passage de paramètre à l'exécution d'un conteneur

- La principale méthode pour passer des paramètres à un démarrage de conteneur consiste à utiliser l'option **-e** de la commande **docker run**. Cette option permet d'assigner des variables d'environnement système au conteneur. Par conséquent, les images exécuté doivent être taillé pour exploiter ces variables.
 - Ex : `root@docker-invivo:~# docker run -d -e NGINX_PORT=80 nginx`

Les images Docker

Qu'est-ce qu'une image Docker

Créer une image à partir d'un conteneur

Créer une image à partir d'un Dockerfile

Stocker et récupérer des images depuis le Docker Hub

Mettre en place un registry privé et y stocker ses images

Qu'est-ce qu'une image Docker ?

- Représente la capture d'un système de fichier
- Peut-être associé à une image parente
- Une image sans parent est une image de base
- Plusieurs conteneurs peuvent avoir en commun les mêmes images.

Créer une image à partir d'un conteneur

```
root@docker-invivoo:~# docker run -it --name invivoo-nginx fedora /bin/bash
> yum -y update
> yum -y install nginx
> exit
root@docker-invivoo:~# docker commit invivoo-nginx daniellavoie/invivoo-nginx:0.0.1
sha256:6f9c2a0d85e8b59d59cdddf72ddd00fb0b9d4ae65a672f7c8f47d69086cc7493
root@docker-invivoo:~#
```


Créer ses images à partir d'un Dockerfile

```
FROM java:openjdk-8-alpine
```

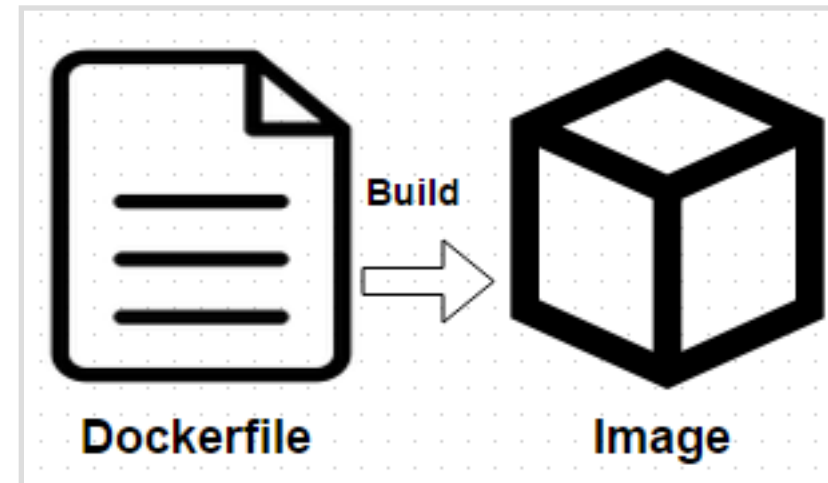
```
MAINTAINER Daniel Lavoie  
<dlavoie@cspinformatique.com>
```

```
EXPOSE 8080
```

```
CMD ["java", "-jar", "/kubik/kubik.jar"]
```

```
VOLUME /kubik/console.log
```

```
ADD target/kubik.jar /kubik/kubik.jar
```



Créer ses images à partir d'un Dockerfile

- Documentation de référence

- <https://docs.docker.com/engine/reference/builder/>

- Quelques dockerfile en exemple

- Mysql : <https://github.com/mysql/mysql-docker/tree/mysql-server/8.0>
- Elasticsearch : <https://github.com/dockerfile/elasticsearch>
- Redis : <https://github.com/dockerfile/redis>

Créer ses images à partir d'un Dockerfile

```
root@docker-invivoo:~# tree -L 2
```

```
.
├── Dockerfile
├── target
│   └── kubik.jar
```

```
root@docker-invivoo:~# docker build . -t daniellavoie/kubik:3.0.0
Sending build context to Docker daemon 148.1 MB
```

```
...
```

```
Successfully built 0673825170cc
```

- Documentation de référence de la commande build
 - <https://docs.docker.com/engine/reference/commandline/build/>

Stocker et récupérer des images depuis le Docker Hub

- Le Docker Hub est un dépôt gratuit pour des images publiques et payant pour des images privés.
- Permet de gérer des builds automatique d'image Docker depuis un repository Github ou Bitbucket
- Gestion par organisation



Charger une image depuis le Docker Hub

```
root@docker-invivo:~# docker pull daniellavoie/kubik:3.0.0
```

- La commande pull permet mettre à jour les images local par rapports aux images présent sur le Docker Hub.
- Lors d'une commande **docker run**, le docker-engine s'occupera de chercher l'image sur les repository disponible dans le cas où l'image n'existerait pas localement.
- Le Docker Hub est le repository par défaut. Il est possible de configurer des repository alternatif.

Charger une image vers le Docker Hub

```
root@docker-invivoo:~# docker push daniellavoie/kubik:3.0.0
The push refers to a repository [docker.io/daniellavoie/kubik]
24e2705dfec4: Pushed
e641266264e1: Layer already exists
bef6fa0c97dd: Layer already exists
da07d9b32b00: Layer already exists
7cbcbac42c44: Layer already exists
```

- Le Docker Hub correspond au repository par défaut.
- Il est tout à fait possible de surcharger et configurer son propre repository privé.
- La plupart des vendeurs fournissent des images officielle de leurs solutions hébergé dans des « Trusted Repositories » sur le Docker Hub.

[Repos](#)
[Stars](#)


daniellavoie

Daniel Lavoie

Paris, France

<http://github.com/daniellavoie/>...

Joined July 2014



[daniellavoie/satisfaction](#)
public

0
STARS

41
PULLS

>
DETAILS



[daniellavoie/invivo-config-server](#)
public

0
STARS

8
PULLS

>
DETAILS



[daniellavoie/centos-beats](#)
public

0
STARS

4
PULLS

>
DETAILS



[daniellavoie/elasticsearch](#)
public

0
STARS

4
PULLS

>
DETAILS



[daniellavoie/bouncer](#)
public

0
STARS

2
PULLS

>
DETAILS

Mettre en place un registry privé et y stocker ses images

- Lancement du registry

```
$ docker run -d -p 5000:5000 --restart=always --name registry registry:2
```

- Stockage d'une image sur le registre

```
$ docker push localhost:5000/ubuntu
```

- Documentation de référence sur la mise sur pied d'un registre :

- <https://docs.docker.com/registry/deploying/>

Le réseau avec Docker

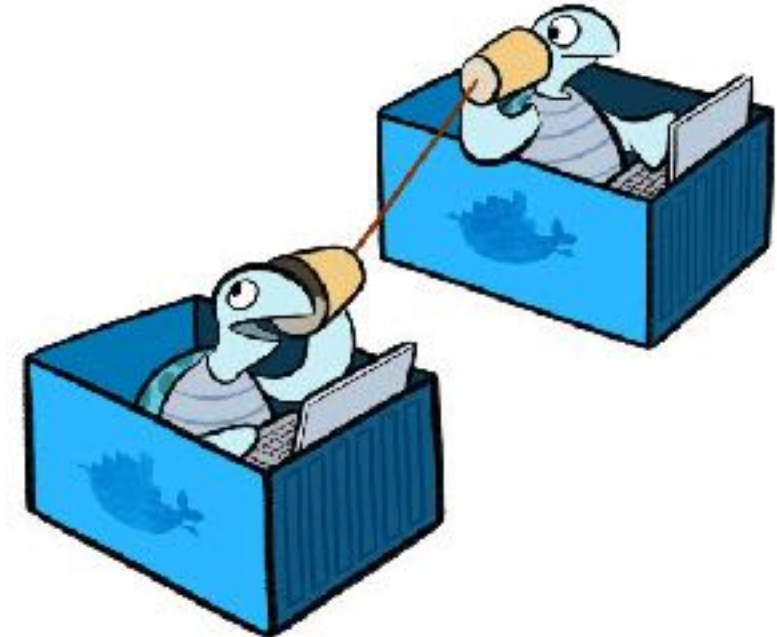
Comprendre la stack réseau de Docker

Utiliser les links Docker

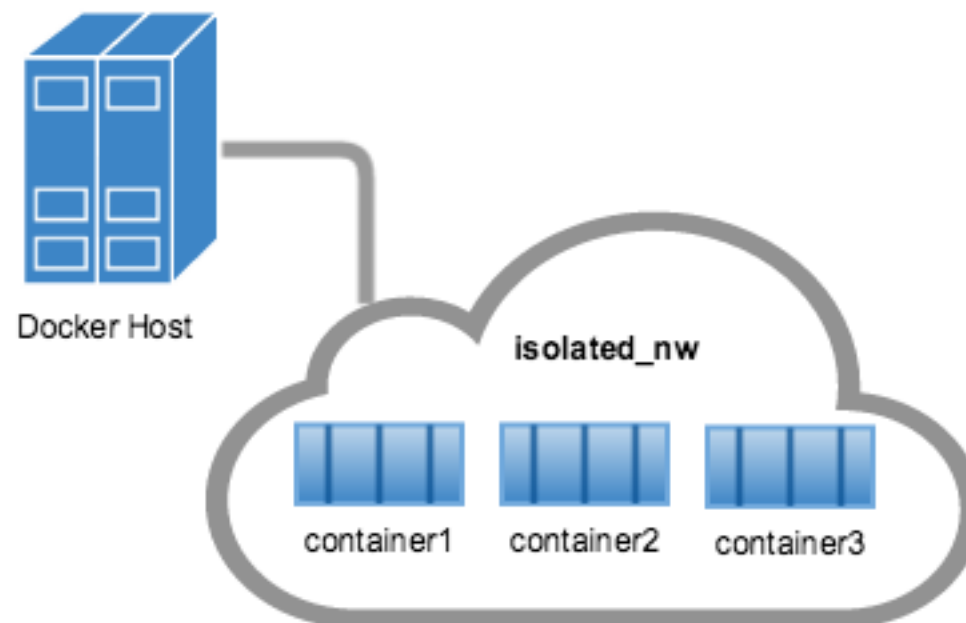
Créer des networks Docker et connaître les drivers réseaux

Comprendre la stack réseau de Docker

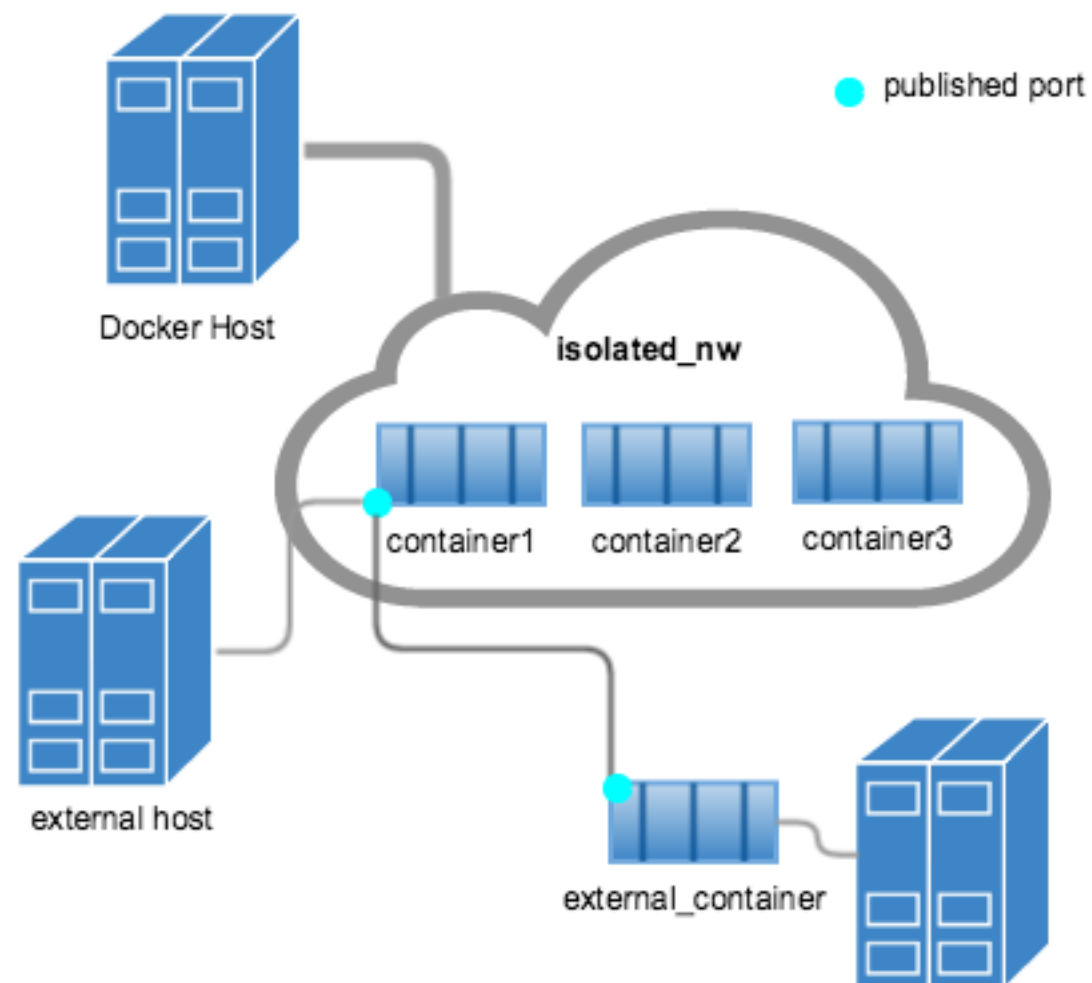
- Les conteneurs sont par défaut isolés les uns des autres.
- Les **Docker Network** permettent à certains conteneurs de communiquer entre eux.
- Chaque network représente un groupe de conteneurs ayant des interfaces réseau pouvant communiquer ensemble.
- Le Docker Engine s'occupe de créer les interfaces réseau en fonction des networks associés aux conteneurs



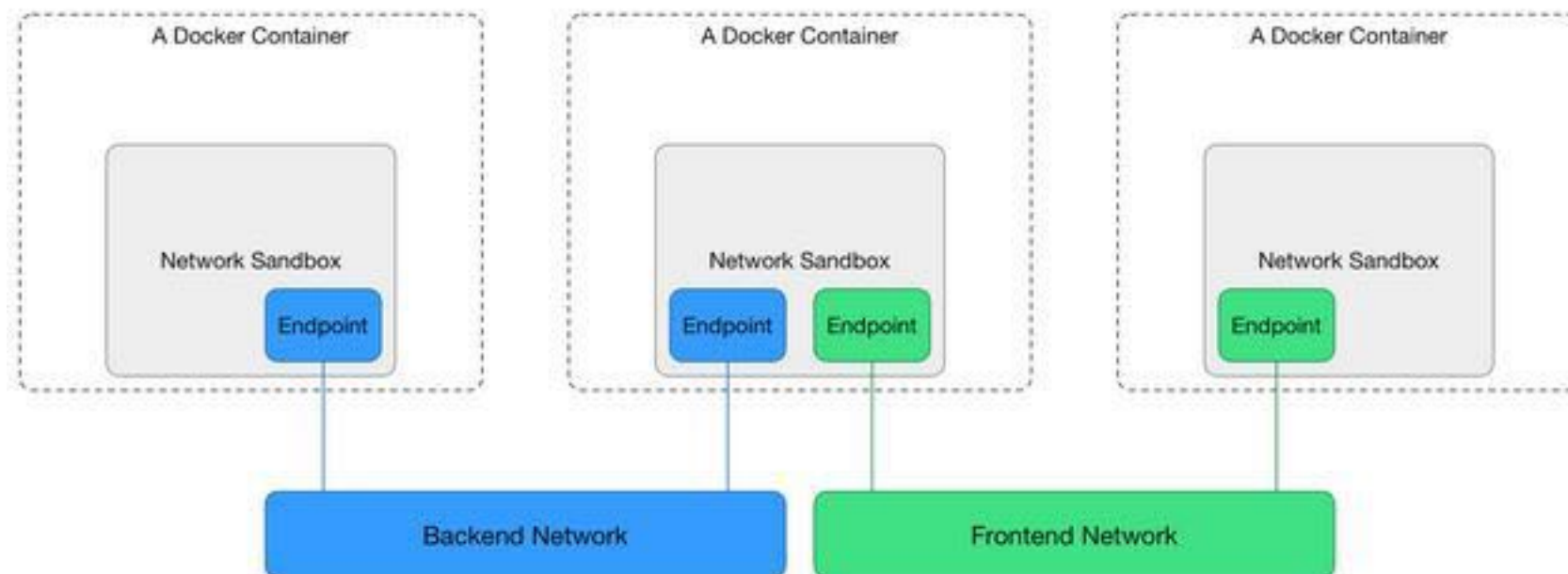
Docker Network - Réseau isolé



Docker Network - Réseau avec ports exposés



Docker Network - Réseau à la carte



Une ligne de commande dédié !

- La ligne de commande **docker network** offrent plusieurs options pouvant permettre d'interagir avec networks géré par le Docker Engine.
- Il ne faut jamais oublié que le Docker Engine n'hésitera jamais à créer les network dont il a besoin. La commande **docker network** sert donc principale pour débbugger et pour nettoyer les network qui ne sont plus actif.

```
root@docker-invivoo:~# docker network ls
NETWORK ID          NAME                DRIVER              SCOPE
04f8d0457c4c        bridge             bridge              local
418a1ab4067e        host               host                local
6b07f6c454d6        traefik_default    bridge              local
69f6cdcb1dc7        none               null                local
```

Option de docker network

- Affichage des network

```
$ docker network ls
```

- Connexion d'un conteneur à un network

```
$ docker network connect new_network conteneur
```

- Création d'un réseau

```
$ docker network create new_network
```

- Suppression d'un réseau

```
$ docker network rm new_network
```

Option de docker network

- Inspection d'un network

```
$ docker network inspect new_network
```

- Déconnexion d'un conteneur d'un network

```
$ docker network new_network conteneur
```


Utiliser les links Docker

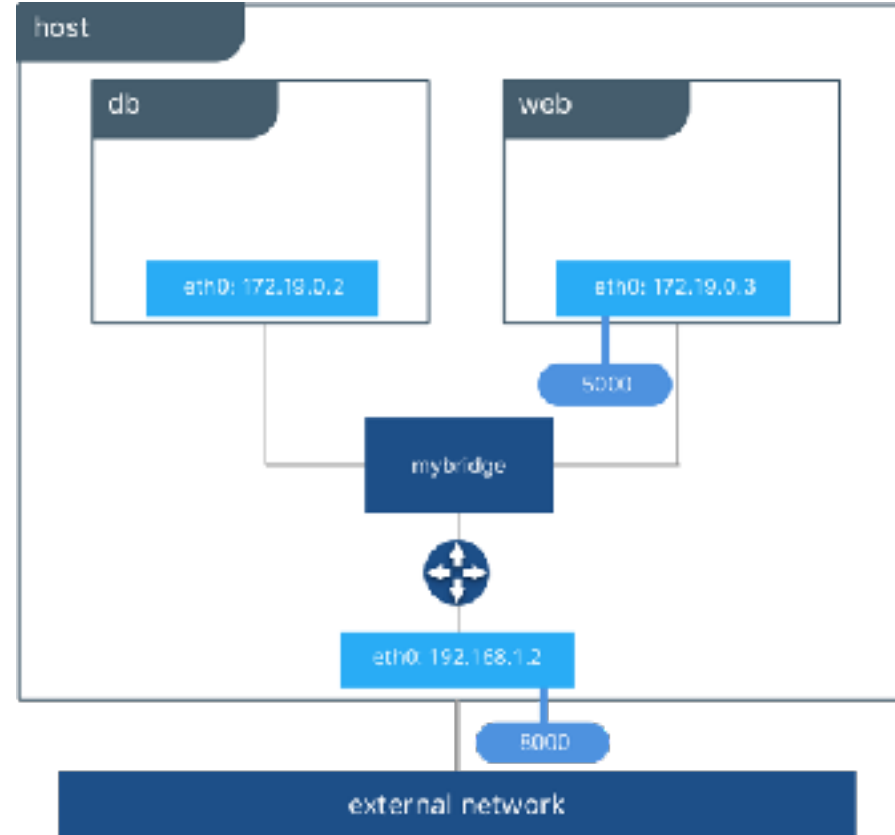


Les drivers réseaux

- **3 Type des drivers**
 - bridge
 - overlay
 - macvlan

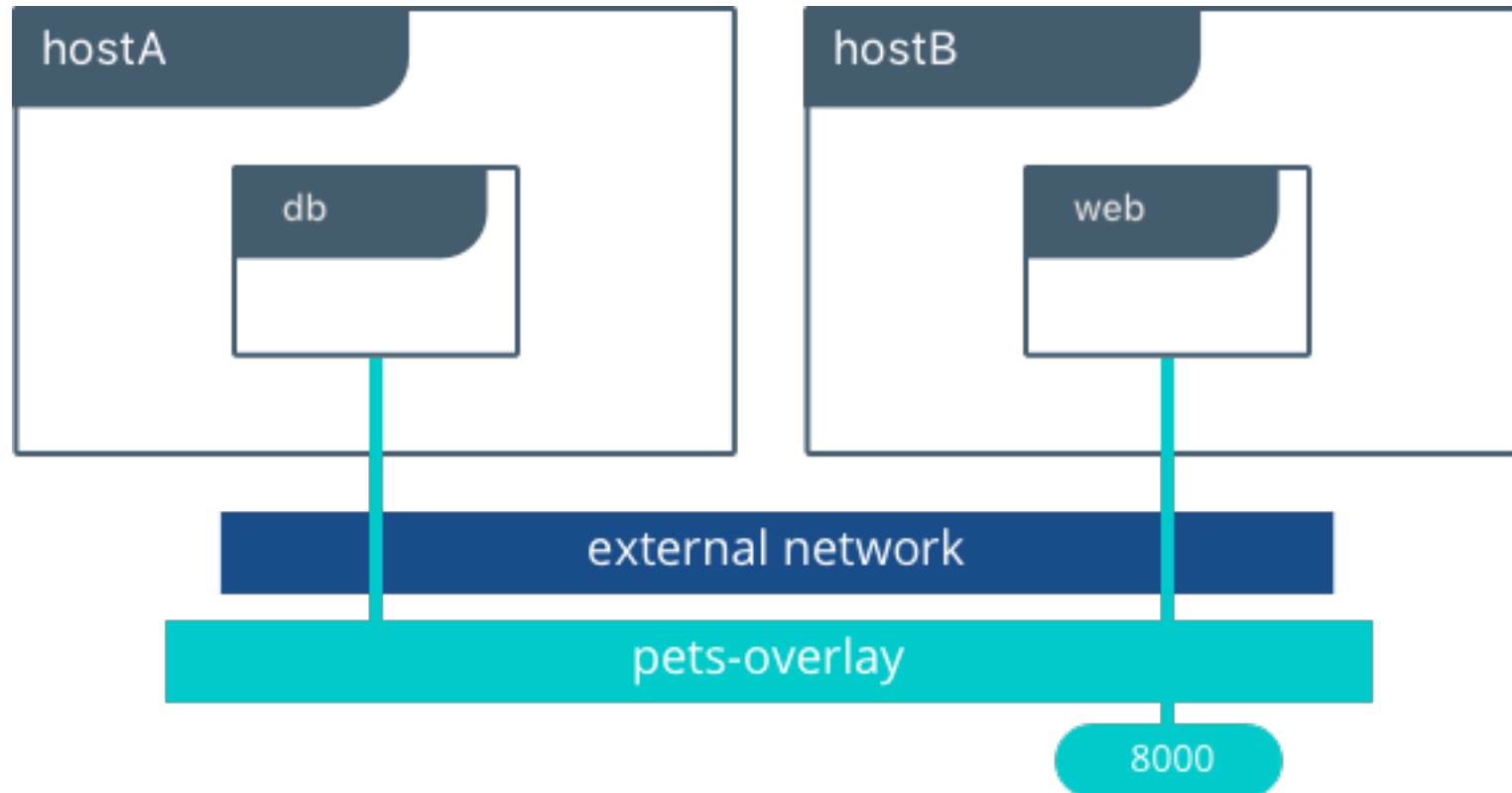
Bridge Driver

```
$ docker network create -d bridge mybridge
$ docker run -d --net mybridge --name db redis
$ docker run -d --net mybridge -e DB=db -p 8000:5000 --name web chrch/web
```

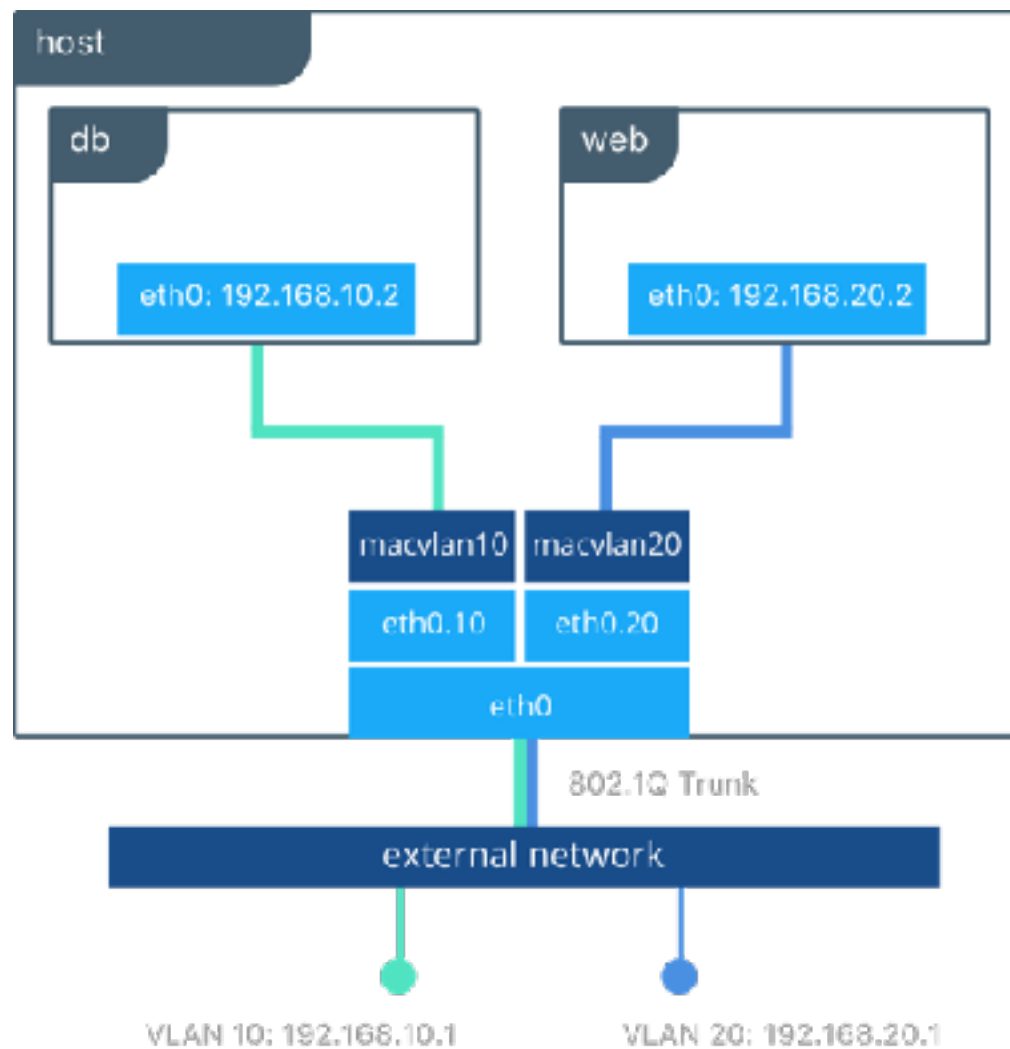


Overlay Driver

```
$ docker network create -d overlay --opt encrypted pets-overlay
$ docker service create --network pets-overlay --name db redis
$ docker service create --network pets-overlay -p 8000:5000 -e DB=db --name web chrch/web
```



MACVLAN Driver (pour les sysadmin)



Pour aller plus sur le sujet

<http://blog.nigelpoulton.com/demystifying-docker-overlay-networking/>

La persistance des données avec Docker

Présentation des volumes Docker

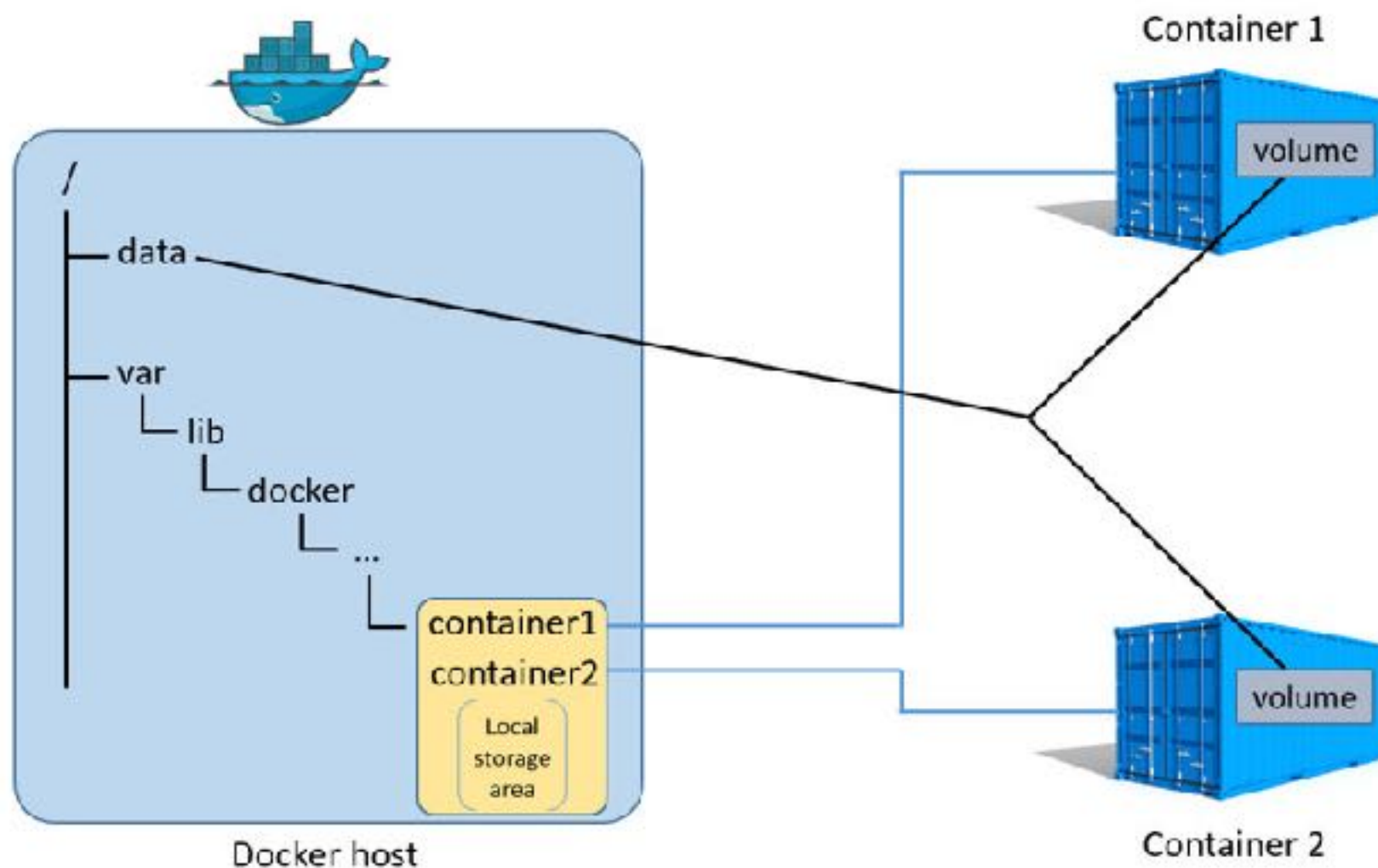
Créer et persister des volumes Docker (host/conteneur, inter-conteneurs)

Data Volume Container

Présentation des volumes Docker

- Les images sont construites sur le Union File System. Les volumes contournent complètement ce système de fichiers.
- Les volumes sont initialisés à la création d'un conteneur.
- Les données du volume écraseront celles de l'image.
- Les volumes peuvent être réutilisés et partagés entre différents conteneurs.
- Les modifications apportées aux fichiers d'un volume sont persistées même si le conteneur est détruit.
- Les volumes sont détruits exclusivement par des opérations manuelles. Le Docker Engine ne supprimera jamais de volume.

Présentation des volumes Docker



Créer et persister des volumes Docker

- Création d'un volume au démarrage d'un conteneur

```
$ docker run -d -P --name web -v /webapp training/webapp python app.py
```

- Création manuelle d'un volume

```
$ docker volume create
```

- Affichage de tous les volumes

```
$ docker volume ls
```

- Inspection d'un volume

```
$ docker volume inspect c4c59f31ebf36...
```

- Suppression d'un volume

```
$ docker volume rm c4c59f31ebf36...
```

Utilisation des Data Volume Container

- Une des bonnes pratiques concernant les volume consiste à créer un conteneur dédié au volume.
- Ceci permet de partager un même volume entre plus
- Cette technique offre aussi l'avantage d'identifier le volume plus facilement.
- L'utilisation d'un conteneur de volume simplifie aussi la gestion de backup.

Utilisation des Data Volume Container

- Création d'un conteneur de volume pour postgres

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

- Lancement d'un conteneur postgres utilisant le volume

```
$ docker run -d --volumes-from dbstore --name db1 training/postgres
```

- Lancement d'un autre conteneur utilisant le volume

```
$ docker run -d --volumes-from dbstore --name db2 training/postgres
```

L'écosystème Docker

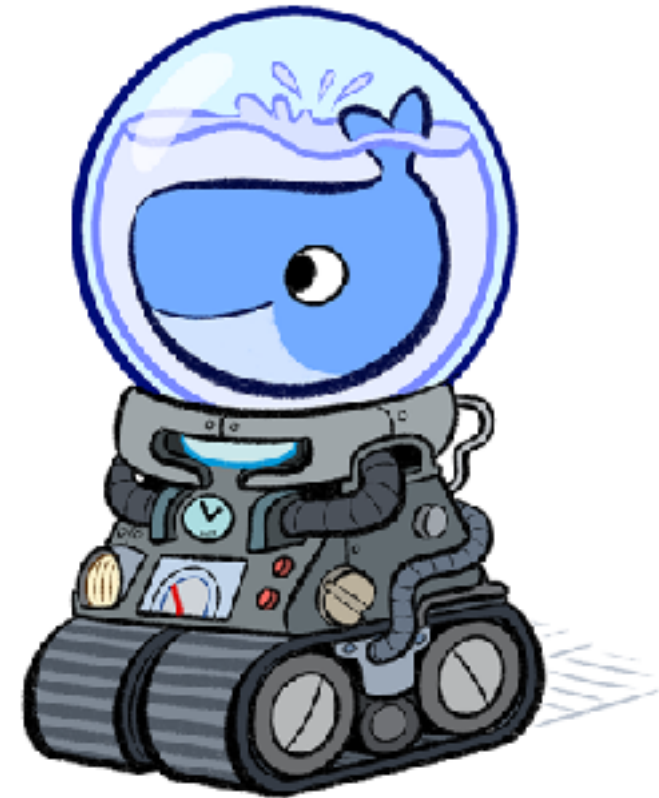
Créer des instances Docker avec Docker Machine

Créer sa stack logicielle avec Docker Compose

Orchestrer le déploiement de conteneurs sur plusieurs machines avec Docker Swarm

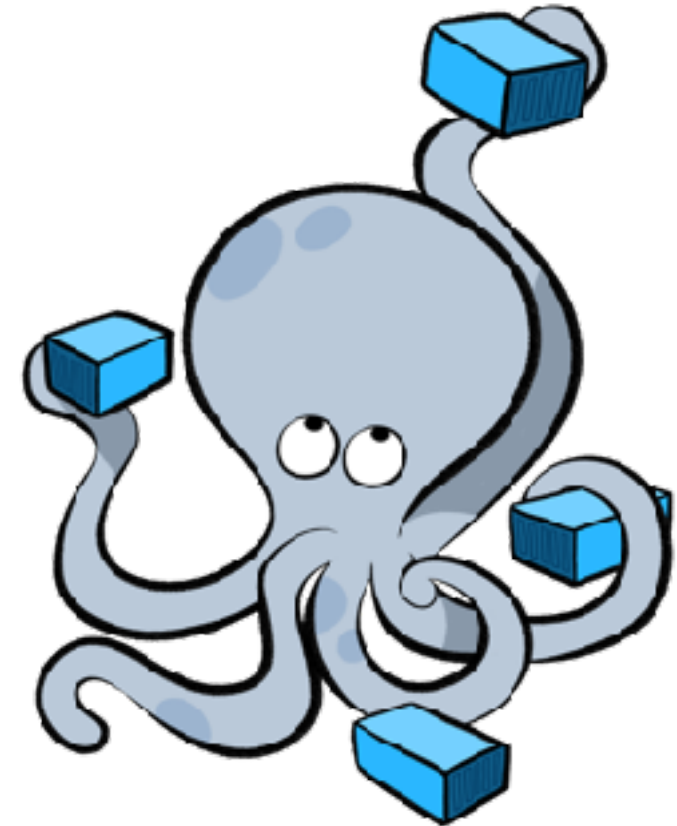
Créer des instances Docker avec Docker Machine

- Ancêtre spirituel de **boot2docker**
- Cas d'utilisations :
 - Installer Docker sur une vieille version de Windows ou OSX
 - Provisionner des hosts Docker sur des serveurs distants
- Les hosts provisionnés peuvent être :
 - Physique
 - Virtuel en local
 - Sur un provider de cloud



Créer sa stack logicielle avec Docker Compose

- Docker Compose permet de configurer un environnement de conteneurs à partir d'un fichier **docker-compose.yml**.
- Toutes les options de la commande **docker run** sont configurable dans le fichier **docker-compose.yml**.
- Idéal pour créer des environnements à la demande pour le développement et les tests automatisé de la software factory.



Configuration As Code ?

Quecéça ?



Une exemple de fichier Docker Compose

```
version: '2'
services:
  web:
    build: .
    ports:
      - "5000:5000"
    volumes:
      - ./code
      - logvolume01:/var/log
    links:
      - redis
  redis:
    image: redis
volumes:
  logvolume01: {}
```

Exemple de paramétrage par Docker Compose

- **command** : Surcharge la commande par défaut du conteneur
- **image** : Image à utiliser pour le conteneur
- **environment** : Variables d'environnement à appliquer aux conteneur
- **ports** : Listes des ports à exposer pour le conteneur
- **networks** : Liste des Docker Network auquel app
- **volumes** : Déclaration des volumes du conteneur
- **volumes_from** : Liste des conteneurs de volumes

Commandes de docker-compose

- **Lancement d'un docker compose en background**

```
$ docker-compose up -d
Creating network "wordpress_default" with the default driver
Creating wordpress_db_1
Creating wordpress_wordpress_1
$
```

- **Arrêt de l'environnement**

```
$ docker-compose stop
Stopping wordpress_wordpress_1 ... done
Stopping wordpress_db_1 ... done
$
```

Commandes de docker-compose

- Nettoyage d'un environnement

```
$ daniellavoie$ docker-compose down
Removing wordpress_wordpress_1 ... done
Removing wordpress_db_1 ... done
Removing network wordpress_default
$
```

- Mise à jour des services modifiés

```
$ daniellavoie$ docker-compose up -d
wordpress_db_1 is up-to-date
Recreating wordpress_wordpress_1
```

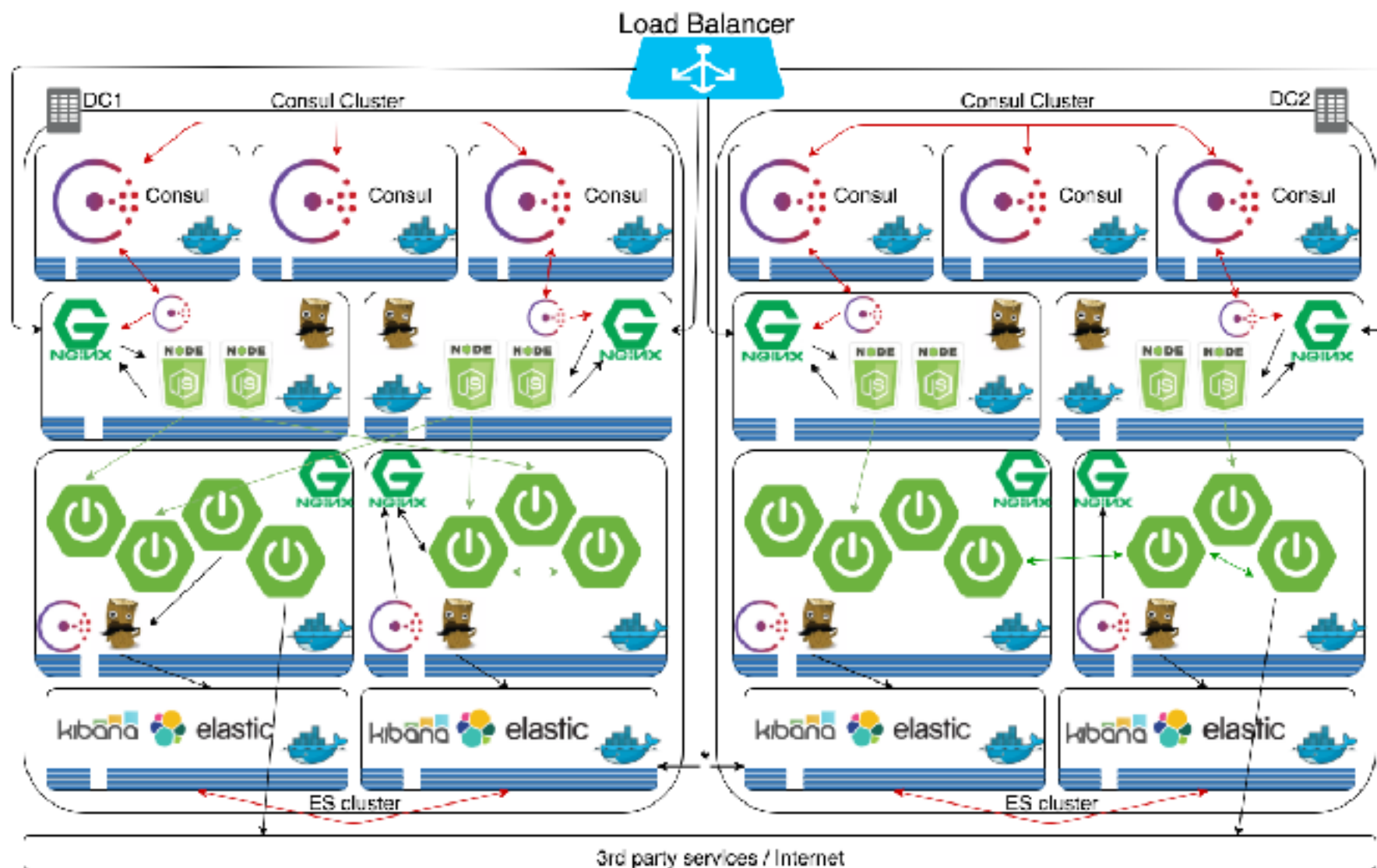
Concepts avancés

Mettre en place une architecture microservices avec
Docker

Sécuriser son infrastructure Docker (TLS, App Armor,
SELinux...)

Docker in Docker

Mettre en place une architecture microservices avec Docker



Sécuriser son infrastructure Docker

De par son utilisation des namespaces et des Cgroup Linux, Docker est nativement sécuritaire. Cependant, il est possible d'aller encore plus loin. Un petit aperçu des possibilités :

<https://docs.docker.com/engine/security/security/>

Docker in Docker

Dockerception !



Et pourquoi du Dockerception ?

Une seule raison valable !

...

Builder des images Docker depuis un Jenkins déployé
dans un conteneur



One Docker File To Build Them All

```
FROM jenkins

# Switch user to root so that we can install apps (jenkins image switches to user "jenkins" in the end)
USER root

# Install Docker prerequisites
RUN apt-get update -qq && apt-get install -qqy \
    apt-transport-https \
    apparmor \
    ca-certificates \
    lxc \
    supervisor

# Install Docker from Docker Inc. repositories.
RUN echo deb https://apt.dockerproject.org/repo debian-jessie main > /etc/apt/sources.list.d/docker.list \
    && apt-key adv --keyserver hkp://p80.pool.sks-keyservers.net:80 --recv-keys \
    58118E89F3A912897C070ADBF76221572C52609D \
    && apt-get update -qq \
    && apt-get install -qqy docker-engine
```

Plus d'informations

Restez connecté

www.invivoo.com

www.xcomponent.com

www.itfinancialnews.com



13, Rue de l'Abreuvoir
92 400 Courbevoie
France

+33 1 80 88 70 00

solutions@invivoo.com

Landsdowne House / City Forum
250 City Road - London EC1V 2PU
United Kingdom

+44 203 695 1784

 [@Invivoo](https://twitter.com/Invivoo)
Page corporate

 [Invivoo](https://www.linkedin.com/company/invivoo)
Page entreprise

 [Invivoo](https://www.facebook.com/Invivoo)
Page interne

Multi-assets
Tempo
réc
Multi-assets
FULL STACK
Systèmes critiques
FLOW BUSINESS
ENABLER
CLIENTS EXIGEANTS
CHANGER LE MONDE
Cloud Grid
Computing
AGILITE
Business Intelligence
COMPOS "TOUT
ENGAGEMENT