

# DOCKER MEETUP



Christophe Labouisse / @XtlCnslt

# #ME, #MYSELF AND #I

## CHRISTOPHE LABOISSE



- ➔ Développeur Freelance
- ➔ Java mais pas que ...
- ➔ Côté front : Angular, Ionic
- ➔ Sous le capot : Linux, Docker

# DOCKER @ HOME

Retour sur l'installation & l'utilisation de Docker sur une machine dédiée

Docker est souvent associé à des déploiements Cloud

# POURQUOI ?

Utiliser une infrastructure existante

Démonstration, transition plus simple

Amusant & permet de mieux comprendre Docker

# POINT DE DÉPART

## DÉDIBOX

Quad-core, 16Go de RAM, 2To de disque

## UTILISATION

Experimentation : Code Story, NoSQL, etc.

Serveur Minecraft

Supervision avec Zabbix

# VIRTUAL MACHINES

## LIBVIRT SUR DEBIAN STABLE

- + Ça marche bien
- Consommateur en resources
- Pas super flexible
- Installation *compliquée*

# ÇA MARCHE BIEN

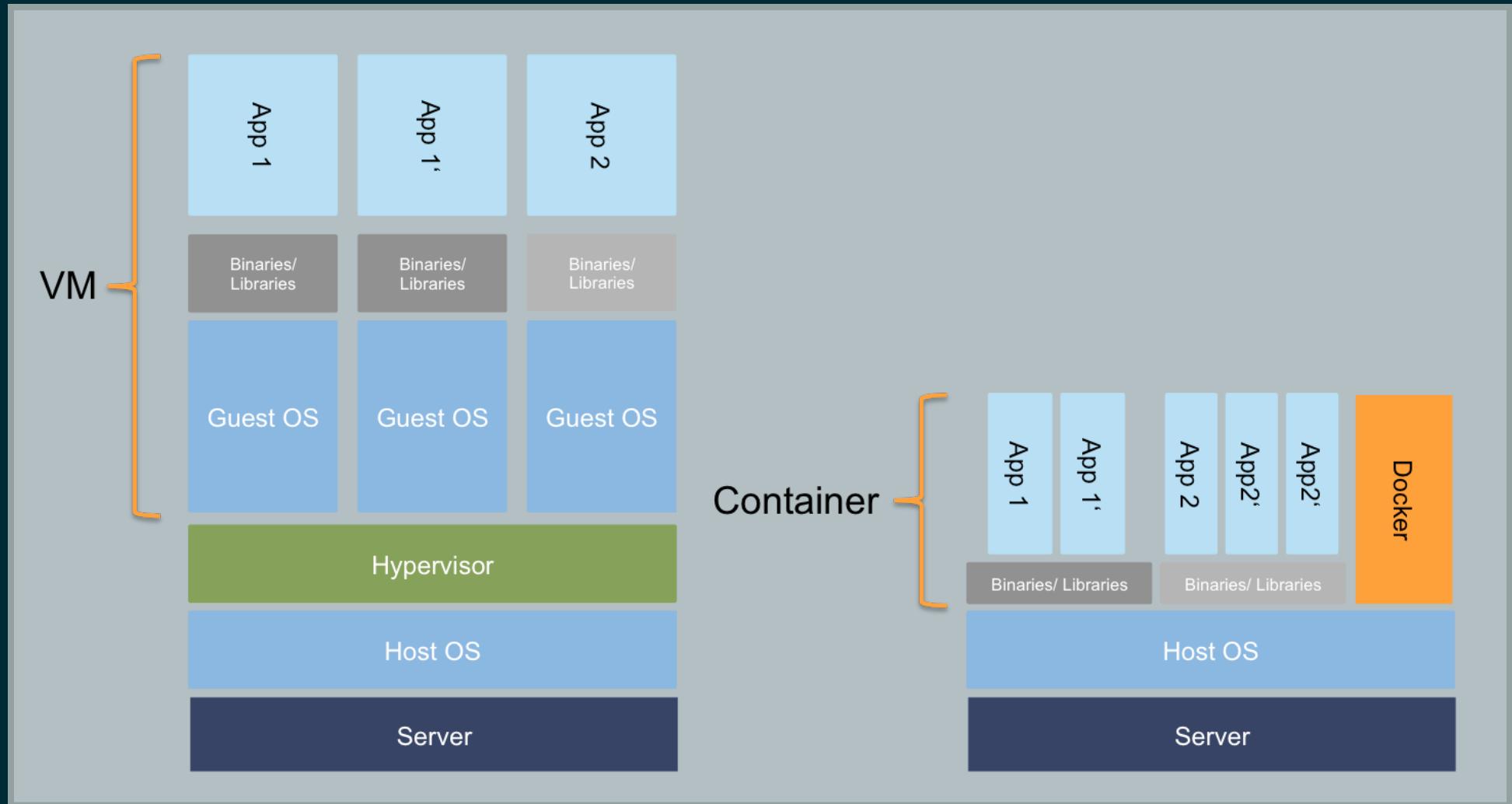
Contrôle total des ressources allouées à la VM

Indépendance totale entre l'OS de l'host et des VM

Isolation entre les VM

Bonnes performances grâce à kvm

# VM VS CONTAINERS



# **FLEXIBILITÉ**

Allocation statique des ressources

Granularité par VM et non par service

# **CONFIGURATION**

Fichier de configuration très (trop) complet décrivant toute la machine virtuelle

Installation d'un OS complet pour chaque VM

# INSTALLATION DOCKER

# PLAN DE BATAILLE

- ➔ Tout sauvegarder
- ➔ Réinstallation à partir de zéro
- ➔ Installer & configurer Docker
- ➔ Tout sera en container
- ➔ Minecraft doit remarcher en 24 heures

# INSTALLATION

- ➔ Installation à partir de l'image Ubuntu 14.04 LTS
- ➔ La version de Docker d'Ubuntu n'est pas satisfaisante
  - Elle renomme tout en `docker.io`
  - Version ancienne (1.0.1 actuellement)
- ➔ Doc d'installation : <http://xlct.it/1C43RAg>
- ➔ Installation à partir des binaires de Docker

```
curl -ssl https://get.docker.com/ubuntu/ | sudo sh
```

# POST INSTALLATION

- ➔ création d'un groupe docker
- ➔ configuration de ufw (firewall)
- ➔ activation du *memory & swap accounting*

# VICTOIRE

```
> docker run --rm -ti busybox echo "Yeeeees"  
Yeeeees
```

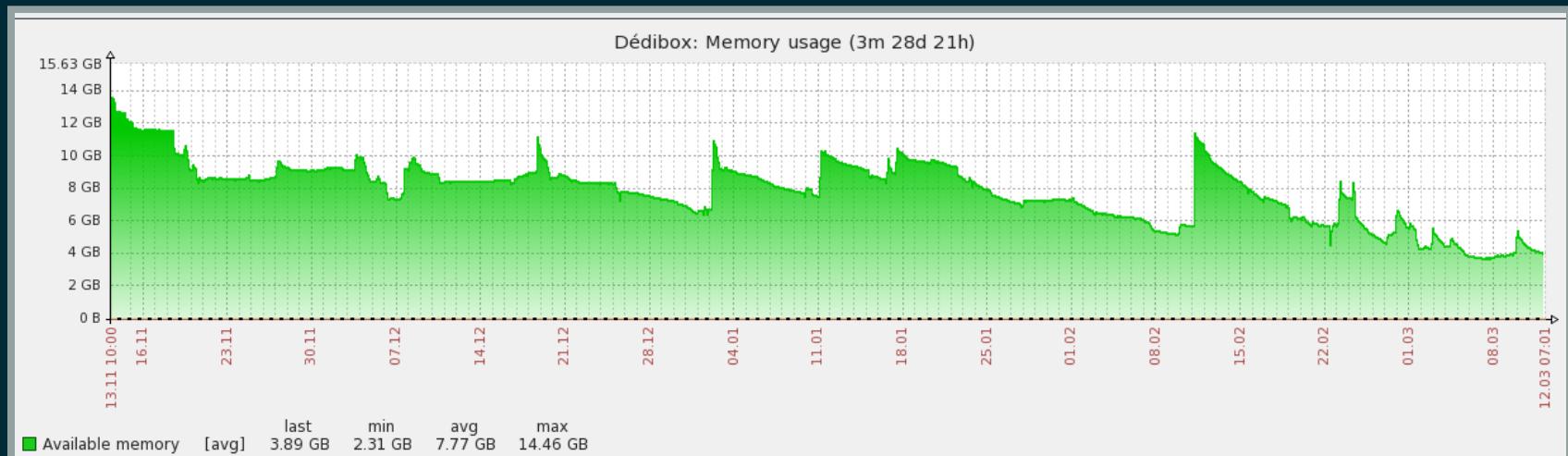
# LES IMAGES

- ➔ Google, Github & Docker Hub sont nos amis
- ➔ Un container = un service
- ➔ 90% d'images personnalisées

# BILAN PARTIEL

# MEILLEUR UTILISATION DES RESOURCES

- ➔ Un vingtaine de containers tournant en continu dont :
  - NoSQL : MongoDB, ElasticSearch (2 serveurs)
  - Jenkins : un serveur et deux esclaves
  - Zabbix : serveur, base de données, web
  - Applications : serveurs Minecraft, appli jHipster
- ➔ Pas de problème de mémoire



LES PROBLÈMES

RAPIDE DÉMO

# ÇA MARCHE BIEN MAIS ...

- ! Plein de process qui tournent avec l'utilisateur root
- ! Accès aux containers
- ! Supervision
- ! Ordonnancement

# PROCESS SOUS ROOT



# PROCESS SOUS *ROOT*

- ➔ Pas super grave
- ➔ Peut-être résolu dans les images
  - `USER xxx`
  - En utilisant un script de démarrage
- ➔ Résolution en cours avec le namespace *user*
  - <https://github.com/docker/docker/issues/7906>
  - <https://github.com/docker/docker/pull/11253>

# SUPERVISION

- ➔ L'outil de supervision est Zabbix mais il en existe d'autres
- ➔ Surveillance globale du host
- ➔ Surveillances des containers

# AVANT



Supervision avec Zabbix : agents sur le host et les VM

# APRÈS



Supervision avec Zabbix : un seul agent sur le host

# SOLUTION NAÏVE

Installer un agent Zabbix dans chaque container

- 👎 À l'opposé du principe un container = un service
- 👎 Pas génial en terme de consommation des ressources
- 👎 Nécessite de n'utiliser *que* des images personnalisées

# RÉCUPÉRATION DES INFORMATIONS

Dans un premier temps on peut partir sur :

- ➔ Le nombre de containers
- ➔ Pour chaque container :
  - ➔ Son adresse IP
  - ➔ Son statut (running, paused, stopped, crashed)
  - ➔ L'utilisation CPU
  - ➔ L'utilisation mémoire
  - ➔ L'activité réseau

# MÉTRIQUES UTILISATEUR DE ZABBIX

- ➔ Zabbix est livré avec de nombreuses métriques pré-définies
- ➔ Il est possible d'ajouter des métriques utilisateur
- ➔ Principe :
  - ➔ L'utilisateur définit un script
  - ➔ Zabbix lance le script *régulièrement*
  - ➔ Le résultat constitue la valeur de la métrique

# NOMBRE DE CONTAINERS

## PETITE DÉMO

On compte facilement le nombre de containers en shell :

- ➔ Le nombre de containers en cours d'exécution :

```
docker ps -q | wc -l
```

- ➔ Le nombre de containers plantés :

```
docker ps -a | grep -v -F 'Exited (0)' | \  
grep -c -F 'Exited ('
```

# LES MÉTRIQUES DES CONTAINERS

## LE SIMPLE ...

La commande magique est :  
`docker inspect container`

- ➔ L'adresse IP du container
- ➔ Son statut

# LE MOINS SIMPLE ...



Jérôme Petazzoni (@jpetazzo)

Article de 2013 : [Gathering LXC Docker Containers Metrics](#)

En résumé : aller fouiller dans  
`/sys/fs/cgroup`

DÉMO ...

AND VOILÀ ...

Ou pas !

 CPU & Mémoire

 Activité réseau

# ACTIVITÉ RÉSEAU

- ➔ Les interfaces réseaux sont les namespaces *network* et non dans les cgroups
- ➔ Plusieurs interfaces par cgroup (lo, eth0, etc.)
- ➔ Les process d'un même cgroup peuvent appartenir à des namespaces différents
- ➔ Récupération en trois étapes :
  - ➔ Localiser un process du container
  - ➔ Créer un lien symbolique
  - ➔ Récupérer les métriques avec `ip netns exec`

# ACTIVITÉ RÉSEAU VERSION MOI

- ➔ Commande docker exec à partir de Docker 1.3
- ➔ On récupère l'activité réseau avec ifconfig eth0

# IMPLEMENTATION



# SCRIPT DE COLLECTE

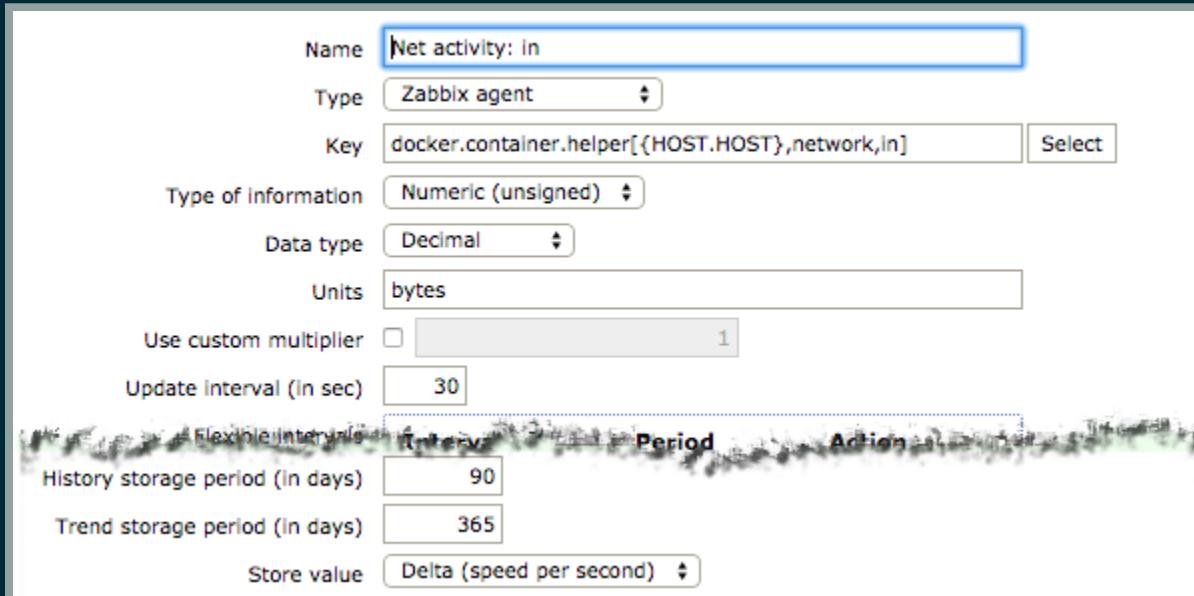
- ➔ Écrit en Python sauf pour le compte des containers
- ➔ Utilise la bibliothèque docker-py

# ZABBIX ITEMS

- ➔ Définir le fichier `/etc/zabbix/zabbix_agentd.conf` d'host:

```
UserParameter=docker.container.count[*],/usr/local/bin/containerCount.sh  
UserParameter=docker.container.helper[*],/usr/local/bin/containerHelper.py
```

- ➔ Déclaration des *items*:



# ZABBIX HOSTS

The screenshot shows the Zabbix host configuration interface. At the top, there are fields for 'Host name' (minecraft-server) and 'Visible name' (Minecraft Server). Below these, under 'Groups', there are two sections: 'In groups' (Virtual machines) and 'Other groups' (Discovered hosts, Hypervisors, Linux servers, Templates, Zabbix servers). A 'New group' input field is also present. In the bottom section, 'Agent interfaces' are configured with an IP address of 172.16.66.1, connect to IP, port 10050, and a default status.

Agent interfaces	IP address	DNS name	Connect to	Port	Default
	172.16.66.1		IP	10050	<input checked="" type="radio"/> Remove

- ! *Host name* doit être exactement le nom du containers
- ! *IP address* est l'adresse IP du host

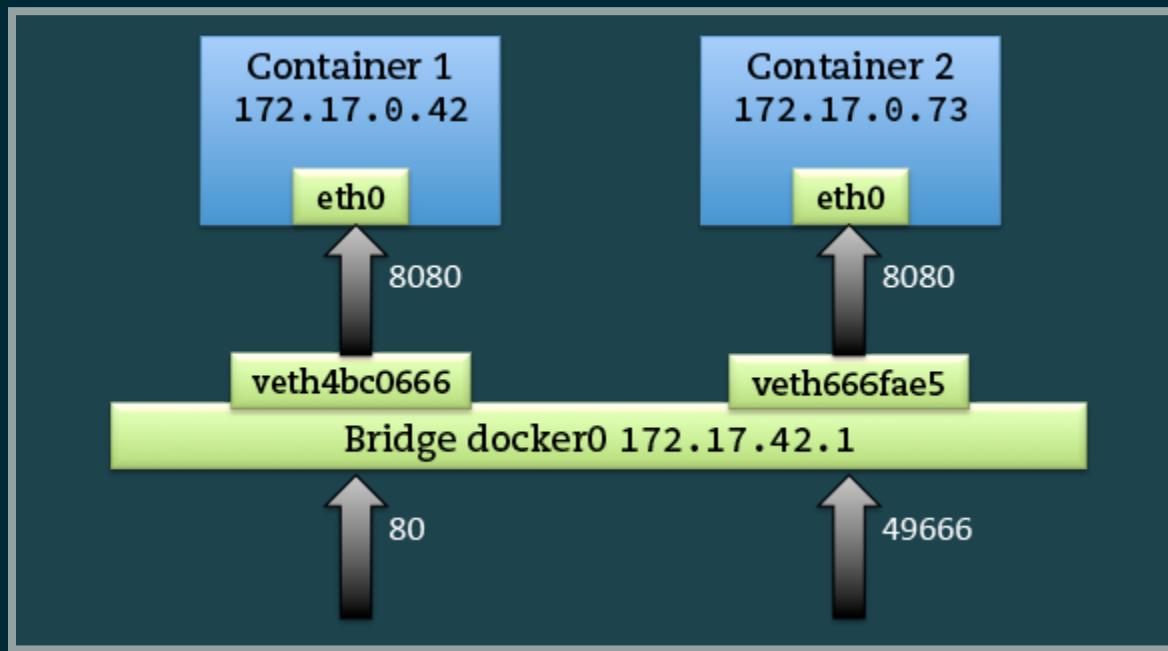
# DOCKER 1.5

- ➔ Docker 1.5 : command `docker stats`
- ➔ Stream l'ensemble des métriques d'un container
- ➔ Plus efficace car on *push* vers Zabbix
- ➔ Script de Tristan Carel

# ACCÈS AUX CONTAINERS

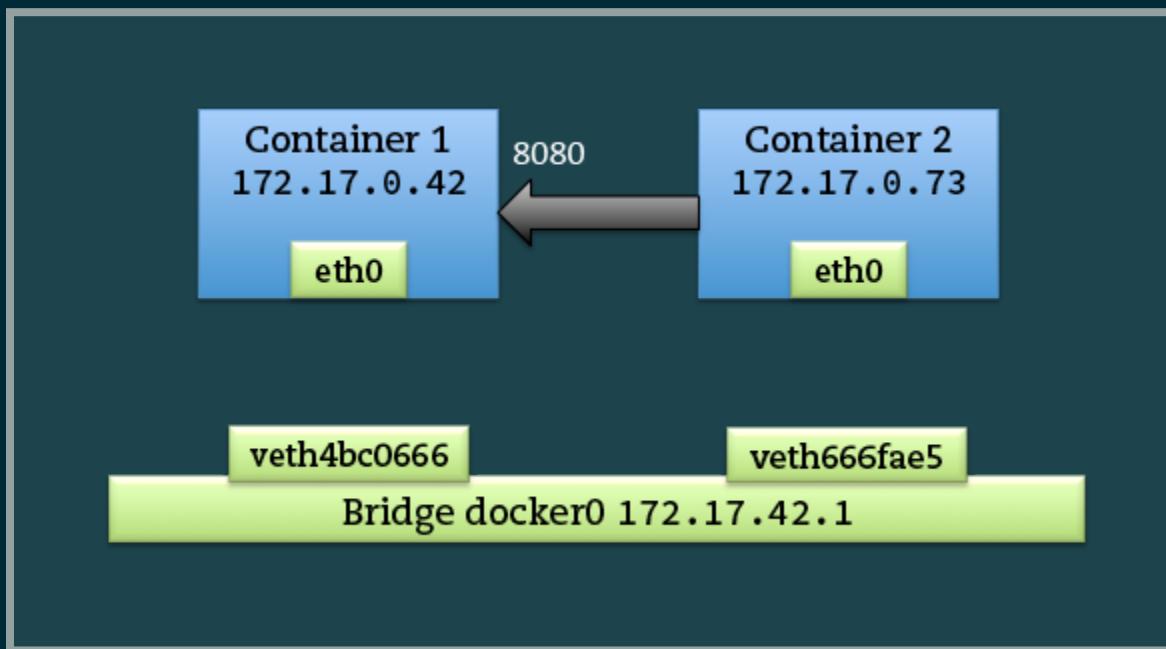
- ➔ On peut publier les ports exposés par un container
- ➔ Les *links* permettent aux containers de communiquer
- ➔ Le host peut aussi accéder directement aux containers

# PUBLIER LES PORTS



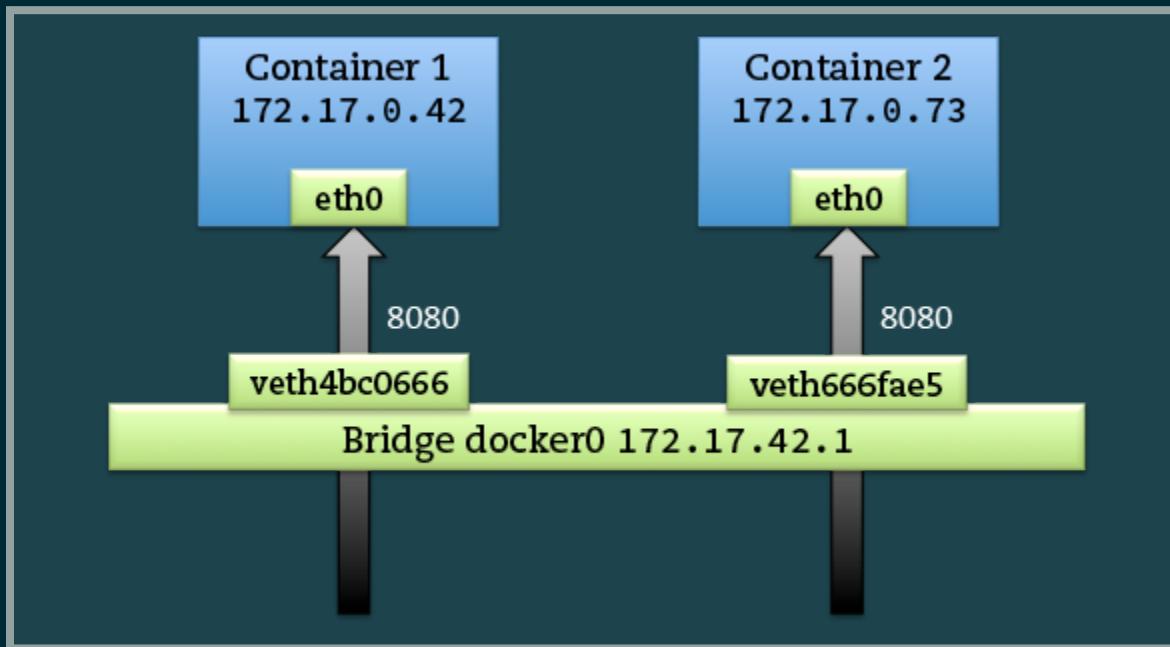
- ➔ Container 1 : `-p80:8080`
- ➔ Container 2 : `-p8080 ou -P`

# LIENS ENTRE CONTAINER



➔ Container 2:--link Container1:service

# ACCÈS DIRECT



- ➔ Pas de configuration
- ➔ Pas de problème pour trouver le n° de port

# PROBLÈME DE L'ACCÈS DIRECT



- ➔ L'adresse IP est alloué au démarrage du container
- ➔ Pas de moyen *simple* de la retrouver

# DNS DYNAMIQUE

## DÉMO

- ➔ On utilise la commande docker events
- ➔ On récupère l'adresse IP
- ➔ Mise à jour de bind

# ORCHESTRATION

# DES PROBLÈMES, TOUJOURS DES PROBLÈMES ...

- ➔ Au (re)démarrage, Docker ne démarre rien par défaut
- ➔ On peut forcer le redémarrage des containers par docker
- ➔ Mais ... pas définition des dépendances
- ➔ En cas d'upgrade d'une image on perd les arguments de création

# DOCKER-COMPOSE (EX FIG)

- ➔ Orchestration *light*
- ➔ Application = plusieurs services
- ➔ Fichier de définition de l'application
- ➔ Commande pour gérer l'application

# FICHIER DE CONFIGURATION

```
server:
  image: jenkins:weekly
  ports:
    - "18080:8080"
  volumes:
    - /data/docker/jenkins/server-1:/var/jenkins_home

swarm:
  image: ggtools/jenkins-swarm-slave-compass
  links:
    - server:jenkins
  command: -username swarm -password 'DoUThink14mStupid' -executors 2 -name swarm

swarmdocker:
  image: ggtools/jenkins-swarm-slave-docker
  links:
    - server:jenkins
  command: -username swarm -password 'DoUThink14mStupid' -executors 2 -name swarm
```

# QUELQUES COMMANDES

- ➔ *builder et lancer l'application :*

```
docker-compose up [-d]
```

- ➔ *Lancer une application sans build :*

```
docker-compose start
```

- ➔ *Et plein d'autres : ps, kill, port, scale, etc.*

# CONCLUSION

Et après ?

- ➔ Docker Swarm
- ➔ Kurbernetes
- ➔ Mesos
- ➔ Cloud Hybride

# RÉFÉRENCES

- <https://github.com/ggtools/docker-tools>
- <https://github.com/dockermeetupsinbordeaux/docker-zabbix-sender>
- <http://xlct.it/dockerathome>

# QUESTIONS ?

