



# DOCKER

Les conteneurs, c'est la vie !



# Qui suis-je ?

## Thomas Saquet

(Non. Pas comme le hobbit.)

✉️ [thomas@solution-libre.fr](mailto:thomas@solution-libre.fr)

🐦 X : [@tsaquet](https://twitter.com/@tsaquet) / [@qtgeekes](https://twitter.com/@qtgeekes)

DMETHOD : [BlueSky : @tsaquet.bsky.social](https://blue.sky/@tsaquet)

.twitch : [qtgeekes](#)

▶️ Youtube : [Qu'est-ce que tu GEEKes ?](#)

### Formateur / Vulgarisateur

Formation DevOps depuis 2015

Vidéos Youtube depuis 2017

Streaming Twitch depuis 2020

### Expert Dev / DevOps / Linux

Professionnel depuis 2008

Développement, conseil,  
accompagnement



9h00

**Début du cours**  
(Appel Pepal !)

10h30

**Pause**  
(15 minutes)

12h30

**Pause Déjeuner**  
(1 heure)

15h15

**Pause**  
(15 minutes)

17h00

**Fin du cours**



# Objectifs pédagogiques

À l'issue de la formation, vous serez en mesure de :

- Comprendre le positionnement de Docker et des conteneurs
- Manipuler l'interface en ligne de commande de Docker pour créer des conteneurs
- Mettre en œuvre et déployer des applications dans des conteneurs
- Administrer des conteneurs
- Déployer rapidement des applications à l'aide de conteneurs
- Identifier les risques et challenges inhérents à Docker afin d'anticiper les bonnes solutions



# Organisation du cours

 N'hésitez pas à prendre des notes, à réécrire les explications avec vos mots.

 Posez des questions  Le but c'est que vous compreniez tout !

 A la fin, c'est noté !



# Thèmes abordés dans le cours

- Chapitre 1 : Virtualisation, virtualisation légère
  - Dans lequel nous parlerons virtualisation
- Chapitre 2 : Docker, le porte-conteneurs
  - Dans lequel nous aborderons les concepts de base de Docker
- Chapitre 3 : Les commandes principales
  - Dans lequel nous verrons les commandes principales
- Chapitre 4 : Les images
  - Dans lequel nous parlerons du concept d'images Docker
- Chapitre 5 : Le réseau
  - Dans lequel nous verrons le réseau sous Docker
- Chapitre 6 : Les volumes
  - Dans lequel nous verrons la gestion des données avec Docker
- ...



# Thèmes abordés dans le cours - Suite

- Chapitre 7 : Docker compose
  - Dans lequel nous verrons l'outil docker compose
- Chapitre 8 : L'API
  - Dans lequel nous allons tester l'API
- Chapitre 9 : Portainer
  - Dans lequel nous découvrirons Portainer
- Chapitre 10 : Registry
  - Dans lequel nous étudierons le concept de Registry
- Chapitre 11 : Swarm
  - Dans lequel nous verrons les principes d'orchestration
- Chapitre 12 : Administration
  - Dans lequel nous parlerons de l'administration du démon Docker



CHAPITRE 1

# **VIRTUALISATION, VIRTUALISATION LEGERE**



# VIRTUALISATION, VIRTUALISATION LEGERE

- **Virtualisation** nous permettra d'aborder le concept de virtualisation.
- **Les conteneurs** nous expliquera le concept de conteneur.



# **VIRTUALISATION, VIRTUALISATION LEGERE**

Virtualisation  
Les conteneurs



# VIRTUALISATION

## Qu'est-ce que c'est ?

- Remplacer du matériel par du logiciel
- Représentation **virtuelle** de :
  - Serveurs
  - Stockage
  - Réseaux
- **Réduire** les dépenses
  - Être plus **efficace**
  - Être plus **agile**

**Virtualisation**  
Les conteneurs



# VIRTUALISATION

Virtualisation  
Les conteneurs

## Qu'est-ce que c'est ?

- C'est un système informatique **virtuel**
- On peut en exécuter **plusieurs** sur un même serveur
- Une machine virtuelle (**VM**) est **autonome** et indépendante
- Un seul ordinateur peut exécuter **des OS différents**
- Une couche logicielle « **hyperviseur** » permet de séparer les VM de la machine hôte



# VIRTUALISATION

## Quelques propriétés

- Indépendant du matériel
  - N'importe quelle VM sur n'importe quel serveur
- Isolé
  - La gestion des pannes est indépendante des VM
  - Contrôle fin des ressources VM par VM
- Encapsulé
  - On peut enregistrer l'état d'une VM à tout moment
  - On peut copier des VMs : ce sont des fichiers

Virtualisation  
Les conteneurs



# VIRTUALISATION

## Quelques propriétés

- Chaque machine virtuelle voit du matériel sur lequel s'installer
  - Les différentes machines virtuelles communiquent comme des machines physiques
  - Dans l'absolu, du point de vue du système d'exploitation dans la machine virtuelle, il n'y a pas de différence avec une machine physique
- C'est le matériel qui s'occupe de faire l'isolation. Elle est bas niveau et robuste

Virtualisation  
Les conteneurs



# VIRTUALISATION

Virtualisation  
Les conteneurs

## Quelques propriétés

- Le principe de la virtualisation est de présenter le même Processeur que sur la machine hôte
  - Les VM verront un CPU AMD quand j'ai un CPU physique AMD, et un Intel quand c'est le cas
  - Si le CPU des machines virtuelles est différent du matériel physique, on parle alors d'émulation. C'est une technologie différente qui a de gros impacts en terme de performance



# VIRTUALISATION

Virtualisation  
Les conteneurs

## Quelques propriétés

Avec les outils de virtualisation (les hyperviseurs), le matériel annexe est souvent **émulé**

- Émulation des cartes réseaux
- Émulation des cartes sons
- Émulation des cartes contrôleur de stockage

L'impact sur les performances est important.

D'où la disponibilités de « **tools** », des outils qui sont des pilotes pour accéder directement au réseau, stockage, affichage, stockage sans faire d'émulation



# VIRTUALISATION

## Ce qu'on peut virtualiser

- Serveurs
  - Plusieurs systèmes sur un seul serveur
- Réseaux
- Postes de travail

Virtualisation  
Les conteneurs



# VIRTUALISATION

## Avantages

- Réduction des **investissements** matériels
- Diminution des **coûts** d'exploitation
- Réduction des **interruptions** de service
- Accélération du **provisionnement** des applications
- Amélioration de la **continuité** et de la **reprise** d'activité

Virtualisation  
Les conteneurs



# **VIRTUALISATION, VIRTUALISATION LEGERE**

Virtualisation  
**Les conteneurs**



# LES CONTENEURS

## Qu'est-ce que c'est ?

- Technologie qui permet
  - D'assembler
  - Et d'isoler

Des applications avec leurs environnement complet

Virtualisation  
Les conteneurs



# LES CONTENEURS

## Pourquoi les utiliser ?

- DevOps par excellence
  - Les dev se concentrent sur les applis
  - Les ops se concentrent sur l'infra
  - Ils échangent des conteneurs !
- Technologies Open Source
- Relativement simple

Virtualisation  
Les conteneurs



# LES CONTENEURS

Virtualisation  
Les conteneurs

## Est-ce qu'on peut faire ça autrement ?

- **ulimit** est une commande qui permet de fixer des limites sur le système.
  - Définit des limites sur une arborescence de processus
  - Ne permet pas de regrouper des processus en dehors de l'arborescence
  - Ne permet que de contrôler certaines ressources
  - Ne permet pas à un processus d'être dans plusieurs limitations

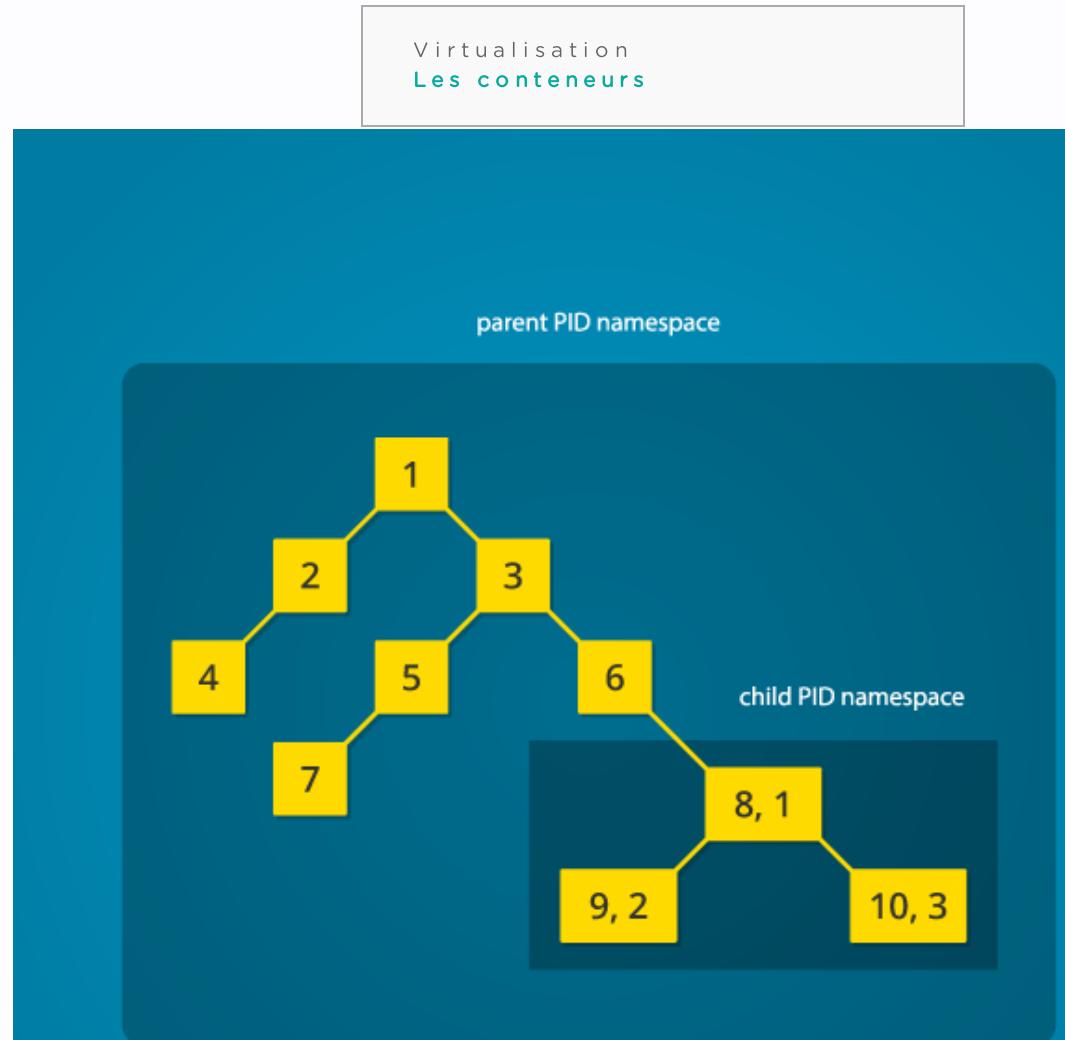


# LES CONTENEURS

## Namespaces

- Isolation d'une hiérarchie de processus
- Dans un namespace, seuls certains processus sont visibles

Virtualisation  
Les conteneurs



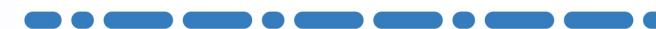


# LES CONTENEURS

## Cgroups

- Limiter l'accès aux ressources CPU et RAM à un ensemble de processus
- Limitation des ressources RAM et CPU des containers.

Virtualisation  
Les conteneurs



# LES CONTENEURS

Virtualisation  
Les conteneurs

## Principes des cgroups

Les cgroups permettent de faire plusieurs choses :

- Limitation de ressources
  - CPU
  - RAM
  - Périphériques de stockage
  - Groupes de périphériques



# VIRTUALISATION, VIRTUALISATION LEGERE

Dans ce chapitre nous avons :

- Vu ce qu'est la virtualisation.
- Appris ce que sont les conteneurs.



## CHAPITRE 2

# **DOCKER, LE PORTE- CONTENEURS**



# DOCKER, LE PORTE-CONTENEURS

- **Docker** nous présentera ce qu'est Docker.
- **Architecture de Docker** nous présentera comment fonctionne Docker et le vocabulaire associé.



# **DOCKER, LE PORTE- CONTENEURS**

**Docker**  
Architecture de Docker



# DOCKER

Docker  
Architecture de Docker

## En quelques mots

L'utilisation de containers a de nombreux avantages :

- **Encapsuler** les applications
- **Virtualisation** légère
- Consommation **faible** de ressources
- Accélération du **déploiement** des applications
- Amélioration de la **portabilité** des applications
- Gestion des ressources **simplifiées**



# DOCKER

Docker  
Architecture de Docker

## Comment ça a commencé

- dotCloud, fondée par Solomon Hykes {Epitech, 2006} en France en 2008 puis relancée à San Francisco en 2010
- Première release open source de Docker par dotCloud en mars 2013
- Déjà en 2014 :
  - 300 000 pulls
  - 18 600 stars sur github
  - 740 contributeurs significatifs
  - 300 projets construits sur Docker
- Des milliers d'applications « Dockerisées »
- Intégré avec... Tout ?



# DOCKER

## Aujourd'hui

- 318 milliards de "**pulls**"
- 7,3 millions d'utilisateurs
- Un écosystème qui continue de croître

(source : <https://www.docker.com/company/>)

**Docker**  
Architecture de Docker



# DOCKER

Docker  
Architecture de Docker

## Pourquoi cet engouement ?

Multiplicité des biens



Les biens doivent-ils interagir ?  
(les bananes à côté des épices)



Puis-je assurer un transport  
rapide et aisément ? (Du bateau au  
train à l'avion)



# DOCKER

Docker  
Architecture de Docker

## Une matrice de l'enfer !

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

# DOCKER

## Solution !

Docker  
Architecture de Docker





# **DOCKER, LE PORTE- CONTENEURS**

Docker  
**Architecture de Docker**



# ARCHITECTURE DE DOCKER

Docker  
Architecture de Docker

## Client - Serveur

L'utilisation de containers a de nombreux avantages :

- Docker Engine : le Docker serveur au centre de l'architecture
- Client : CLI pour interagir avec le serveur
- Des applications avec leurs environnement complet
- Image : image de binaires et de librairies pour exécuter des applications
- Container : une instance d'une image en exécution
- Registry : bibliothèque d'images



# ARCHITECTURE DE DOCKER

## Docker Engine

- C'est le serveur au centre de l'architecture
- Il permet d'exécuter les conteneurs
- C'est un « **daemon** »
- Il utilise les namespaces du noyau Linux et les cgroups
- Les namespaces garantissent l'isolation des conteneurs

Docker  
Architecture de Docker



# ARCHITECTURE DE DOCKER

## Le client

- Le client Docker se connecte au Docker Engine
- Il peut être installé sur la même machine
- Ou sur une machine différente
- Il y a deux types de clients
  - CLI
  - GUI

Docker  
Architecture de Docker



# ARCHITECTURE DE DOCKER

## Images

- Modèles en **lecture seule** utilisés pour créer les conteneurs
- Construites par vous ou d'autres utilisateurs
- Stockées dans **Docker hub** ou votre propre registre
- Basées sur une ou plusieurs images

Docker  
Architecture de Docker



# ARCHITECTURE DE DOCKER

## Docker Hub

- Permet de centraliser les images
- Configuration par défaut
- Permet d'explorer les images
- Permet de récupérer des images
- Permet de stocker ses propres images

Docker  
Architecture de Docker



# ARCHITECTURE DE DOCKER

## Conteneurs

- Espace d'exécution d'application isolé
- Contient tout ce qui est nécessaire à l'exécution
  - Binaires de l'application
  - Bibliothèques requises par l'application

Docker  
Architecture de Docker



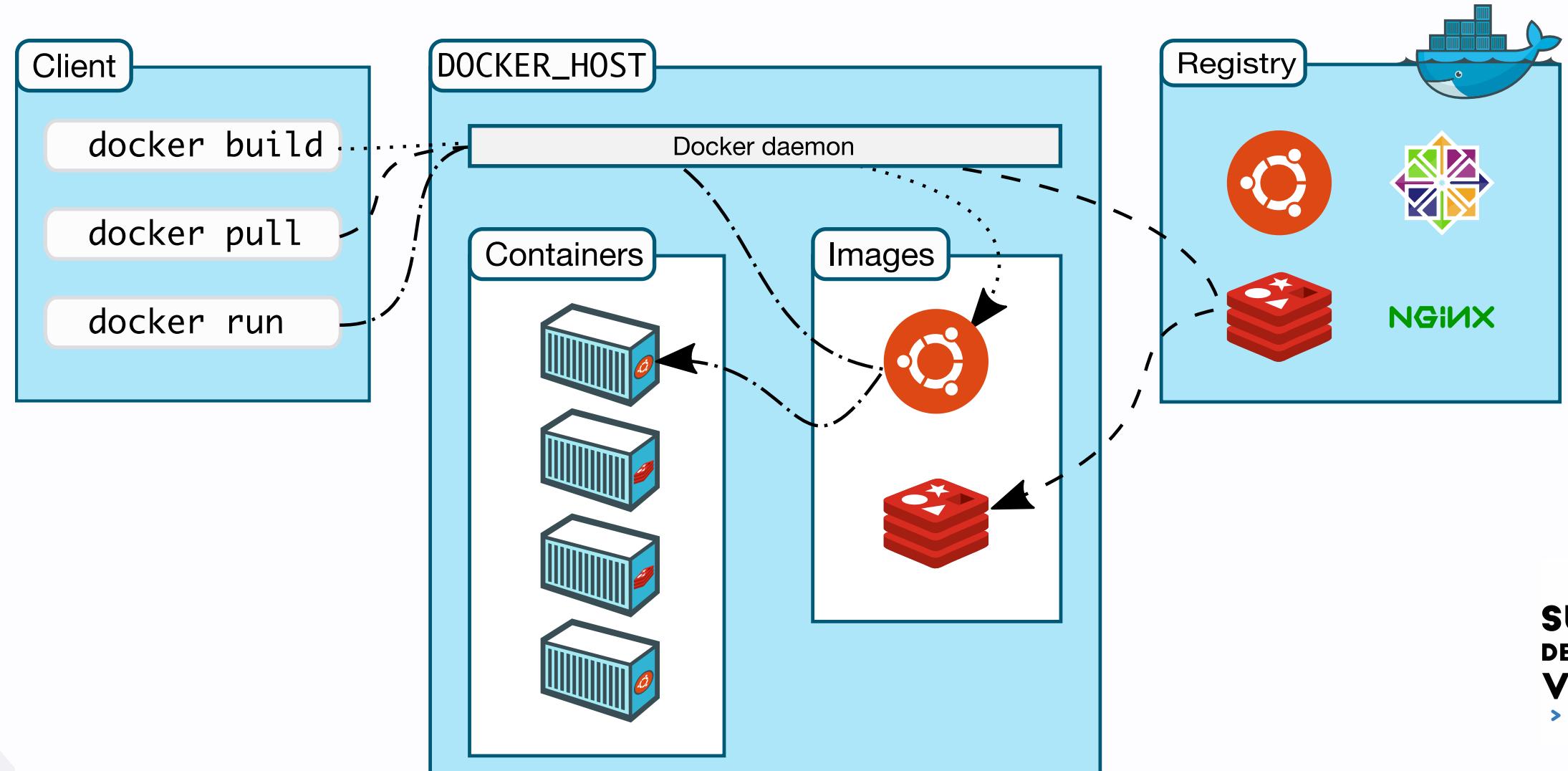
# ARCHITECTURE DE DOCKER

Docker  
Architecture de Docker

## Registre

- Stockage des images
  - Registre local
  - Registre distant
- Accès aux images
  - Registre privé
  - Registre public

# ARCHITECTURE DE DOCKER





# ARCHITECTURE DE DOCKER

## Orchestration

- Automatisation du déploiement
  - Mise à l'échelle
  - Gestion
  - Mise en réseau
- De multiples solutions :
  - Docker swarm mode
  - Docker compose
  - Kubernetes

Docker  
Architecture de Docker



# DOCKER, LE PORTE-CONTENEURS

Dans ce chapitre nous avons :

- Découvert Docker.
- Découvert l'architecture de Docker.



## CHAPITRE 3

# LES COMMANDES PRINCIPALES



# LES COMMANDES PRINCIPALES

- **Docker CLI** nous fera découvrir la CLI de Docker.
- **Docker CLI - TP** nous fera installer et manipuler la CLI.



# LES COMMANDES PRINCIPALES

Docker CLI

Docker CLI - TP



# DOCKER CLI

## Cycle de vie

- Basique :
  - Création à partir d'une image
  - Exécution du processus spécifié
  - Le processus se termine, le conteneur s'arrête
  - Le conteneur est détruit
- Avancé :
  - Création à partir d'une image
  - Exécution du processus spécifié
  - Interagir et effectuer d'autres actions à l'intérieur du conteneur

Docker CLI  
Docker CLI - TP



# DOCKER CLI

Docker CLI

Docker CLI - TP

## Créer et lancer le conteneur

- La « commande run » de docker
  - Crée le conteneur en utilisant l'image spécifiée
  - Exécute le conteneur

```
docker run [options] image [commande] [args]
```



# DOCKER CLI

Docker CLI

Docker CLI - TP

## Lister les conteneurs

### La commande « docker ps »

```
[root@localhost ~]# docker run hello-world
Hello from Docker!
This message shows that your installation appears to be working correctly.

[...]

[root@localhost ~]# docker ps
CONTAINER ID        IMAGE               COMMAND             CREATED            STATUS              PORTS               NAMES
[...]
[root@localhost ~]# docker ps -a
CONTAINER ID        IMAGE               COMMAND             CREATED            STATUS              PORTS               NAMES
0f945d073d93        hello-world        "/hello"           13 seconds ago   Exited (0) 11 seconds ago
25671c7185a1        almalinux          "ping 127.0.0.1 -c 60"   47 hours ago    Exited (0) 47 hours ago
d0ae27b63012        ubuntu:21.04      "bash"              47 hours ago    Exited (0) 47 hours ago
9ecdc7b5cd05        ubuntu:21.04      "bash"              47 hours ago    Exited (0) 47 hours ago
94acff76f0de        ubuntu:14.04      "ps -ef"            47 hours ago    Exited (0) 47 hours ago
db88ee8828bb        ubuntu:14.04      "echo 'hello world'" 47 hours ago    Exited (0) 47 hours ago
64d37bda2457        hello-world        "/hello"           47 hours ago    Exited (0) 47 hours ago
distracted_lamport
great_banzai
blissful_agnesi
naughty_khorana
exciting_ptolemy
pensive_lovelace
serene_bardeen
```



# DOCKER CLI

Docker CLI  
Docker CLI - TP

## Utilisation du terminal

- L'option « -i » indique qu'on souhaite se connecter à stdin
- L'option « -t » permet d'obtenir un terminal

```
[root@localhost ~]# docker run -it ubuntu bash
root@0bcceb898257:/# exit
exit
[root@localhost ~]#
```

- CTRL + P + Q permet de sortir du conteneur sans l'arrêter



# DOCKER CLI

Docker CLI

Docker CLI - TP

- On peut se référer aux conteneurs en utilisant
  - Leur ID (court ou long)
  - Un nom
- L'option « --no-trunc » permet d'obtenir le l'ID long

CONTAINER ID	IMAGE	COMMAND	CREATED	STATUS	PORTS	NAMES
0bcceb8982573b19ba93bb1822592434bd12058a8ace726f307e8c0d2d9e2618	ubuntu	"bash"	6 minutes ago	Exited (0) 6 minutes ago		xenodochial_pascal
c569a0d061deb95abf598d4ce0987ae372a75c5b9aa651911f99116f1a82dac8	ubuntu	"bash"	6 minutes ago	Exited (0) 6 minutes ago		elegant_joliot
0f945d073d9354af8ed38c4ad275363c70a138831a6335a97abc88597408d1bd	hello-world	"/hello"	11 minutes ago	Exited (0) 11 minutes ago		distracted_lamport
1d9fa6ce64f52f04b55ab75b3ca52a12e48a39c0578e6bb67f02d5b35184195a	almalinux	"ping 127.0.0.1 -c 60"	47 hours ago	Exited (0) 47 hours ago		eager_buck
d0ae27b630121e1ba192eeb5a1ec51df192207b14f643efb0b3fb3f878f93226	ubuntu:21.04	"bash"	47 hours ago	Exited (0) 47 hours ago		blissful_agnesi
9ecdc7b5cd059ea312dd46dd45e2744f9ff2580fb3e721777323337e1b4117a8	ubuntu:21.04	"bash"	47 hours ago	Exited (0) 47 hours ago		naughty_khorana
94acff76f0de9ec3d360cfbe69b3656cf53c8a065c58a29a37a78ad10d57f69c	ubuntu:14.04	"ps -ef"	47 hours ago	Exited (0) 47 hours ago		exciting_ptolemy
db88ee8828bb158845fe4c478b0e0042fc7b2c57c919f6be967025f1526150fb	ubuntu:14.04	"echo 'hello world'"	47 hours ago	Exited (0) 47 hours ago		pensive_lovelace
64d37bda2457dd343a30508c60e6e91c691bef1b9bac4af585611baf0efdf539	hello-world	"/hello"	47 hours ago	Exited (0) 47 hours ago		serene_bardeen



# DOCKER CLI

Docker CLI

- Pour lister uniquement les ID courts

```
[root@localhost ~]# docker ps -a -q  
0bcceb898257  
c569a0d061de  
0f945d073d93  
1d9fa6ce64f5  
4835e4554a0e  
b298bed5a0c0  
001b7ba646b8
```

Pour lister le dernier conteneur qui a été lancé

```
[root@localhost ~]# docker ps -l
CONTAINER ID        IMAGE       COMMAND       CREATED          STATUS           PORTS     NAMES
0bcceb898257      ubuntu      "bash"        40 minutes ago   Exited (0) 40 minutes ago

```





# DOCKER CLI

Docker CLI

Docker CLI - TP

- On peut filtrer par rapport au statut du conteneur
  - Par exemple : restarting - running - exited - paused

CONTAINER ID	IMAGE	COMMAND	CREATED	STATUS	PORTS	NAMES
4835e4554a0e	almalinux	"ping 127.0.0.1 -c 6..."	47 hours ago	Exited (2)	47 hours ago	clever_meninsky
b298bed5a0c0	almalinux	"ping 127.0.0.1 -c 6..."	47 hours ago	Exited (2)	47 hours ago	fervent_brahmagupta
001b7ba646b8	almalinux	"ping 127.0.0.1 -c 6..."	47 hours ago	Exited (2)	47 hours ago	thirsty_wright
114e964bb74a	almalinux	"ping 127.0.0.1 -c 6..."	47 hours ago	Exited (2)	47 hours ago	confident_johnson
89ee727e5dcc	almalinux	"ping 127.0.0.1 -c 6..."	47 hours ago	Exited (2)	47 hours ago	busy_bose
8efb561456fc	almalinux	"ping 127.0.0.1 -c 6..."	47 hours ago	Exited (2)	47 hours ago	vibrant_shirley



# DOCKER CLI

Docker CLI  
Docker CLI - TP

## Mode détaché

- Exécution en arrière-plan comme « daemon »
  - Option « -d »
  - On peut lire les logs avec « docker logs »

```
[root@localhost ~]# docker run -d almalinux ping 127.0.0.1
09bde2c57540dcf46ec2de0ed90a1de10224b707e03a520b7a421eb257e336ea
```

```
[root@localhost ~]# docker logs 09bde2c57540dcf46ec2de0ed90a1de10224b707e03a520b7a421eb257e336ea
PING 127.0.0.1 (127.0.0.1) 56(84) bytes of data.
64 bytes from 127.0.0.1: icmp_seq=1 ttl=64 time=0.023 ms
64 bytes from 127.0.0.1: icmp_seq=2 ttl=64 time=0.027 ms
64 bytes from 127.0.0.1: icmp_seq=3 ttl=64 time=0.029 ms
64 bytes from 127.0.0.1: icmp_seq=4 ttl=64 time=0.042 ms
64 bytes from 127.0.0.1: icmp_seq=5 ttl=64 time=0.028 ms
64 bytes from 127.0.0.1: icmp_seq=6 ttl=64 time=0.034 ms
64 bytes from 127.0.0.1: icmp_seq=7 ttl=64 time=0.033 ms
```



# DOCKER CLI

Docker CLI

Docker CLI - TP

## S'attacher à un conteneur

- S'attacher à un conteneur consiste à
  - Ramener l'exécution à l'avant-plan
  - La sortie standard du conteneur revient dans le terminal

```
[root@localhost ~]# docker attach 09bde2c57540dcf46ec2de0ed90a1de10224b707e03a520b7a421eb257e336ea
64 bytes from 127.0.0.1: icmp_seq=199 ttl=64 time=0.026 ms
64 bytes from 127.0.0.1: icmp_seq=200 ttl=64 time=0.042 ms
64 bytes from 127.0.0.1: icmp_seq=201 ttl=64 time=0.042 ms
64 bytes from 127.0.0.1: icmp_seq=202 ttl=64 time=0.027 ms
```



# DOCKER CLI

Docker CLI  
Docker CLI - TP

## Se détacher d'un conteneur

- CTRL + P + Q passe le conteneur en arrière-plan
- Sous conditions :
  - Le conteneur a été démarré avec un terminal
  - L'entrée standard du conteneur est connectée

```
[root@localhost ~]# docker attach 09bde2c57540dcf46ec2de0ed90a1de10224b707e03a520b7a421eb257e336ea
64 bytes from 127.0.0.1: icmp_seq=199 ttl=64 time=0.026 ms
64 bytes from 127.0.0.1: icmp_seq=200 ttl=64 time=0.042 ms
64 bytes from 127.0.0.1: icmp_seq=201 ttl=64 time=0.042 ms
64 bytes from 127.0.0.1: icmp_seq=202 ttl=64 time=0.027 ms

[root@localhost ~]# docker ps
CONTAINER ID        IMAGE               COMMAND                  CREATED             STATUS              PORTS               NAMES
09bde2c57540        almalinux          "ping 127.0.0.1"    19 seconds ago     Up 18 seconds ago   clever_meninsky
```

- CTRL + C interrompt le processus, donc le conteneur



# DOCKER CLI

Docker CLI  
Docker CLI - TP

## Exécuter une commande

- « docker exec » permet d'exécuter un processus supplémentaire à l'intérieur d'un conteneur
- On l'utilise en général pour accéder à la ligne de commande
- Dans ce cas la sortie du terminal n'arrête pas le conteneur

```
[root@localhost ~]# docker run -d almalinux ping 127.0.0.1
617dbe251a6480d53445ca859adb509ead849f2357cba3a5e8ffff65253b4b909
```

```
[root@localhost ~]# docker ps
CONTAINER ID   IMAGE      COMMAND           CREATED          STATUS          PORTS     NAMES
617dbe251a64   almalinux   "ping 127.0.0.1"   5 seconds ago   Up 4 seconds   hopeful_austin
```

```
[root@localhost ~]# docker exec -it 617dbe251a64 /bin/bash
[root@617dbe251a64 /]# ps -ef
UID        PID    PPID  C STIME TTY          TIME CMD
root            1      0  0 15:53 ?        00:00:00 ping 127.0.0.1
root            7      0  0 15:53 pts/0    00:00:00 /bin/bash
root           21      7  0 15:53 pts/0    00:00:00 ps -ef
```



# DOCKER CLI

Docker CLI  
Docker CLI - TP

## Arrêter un conteneur

- « `docker stop` » arrête un conteneur
  - Envoie un signal SIGTERM au processus principal (pid 1)
  - Envoie un SIGKILL au bout d'un moment
  - Cette période d'attente peut être définie avec l'option `-t`
- « `docker kill` » envoie immédiatement un SIGKILL

```
[root@localhost ~]# docker ps
CONTAINER ID        IMAGE               COMMAND             CREATED            STATUS              PORTS          NAMES
617dbe251a64        almalinux          "ping 127.0.0.1"   3 minutes ago    Up 3 minutes           hopeful_austin

[root@localhost ~]# docker stop 617dbe251a64
617dbe251a64

[root@localhost ~]# docker ps -a
CONTAINER ID        IMAGE               COMMAND             CREATED            STATUS              PORTS          NAMES
617dbe251a64        almalinux          "ping 127.0.0.1"   4 minutes ago     Exited (137) 4 seconds ago   hopeful_austin
```



# DOCKER CLI

Docker CLI

Docker CLI - TP

## Redémarrer un conteneur

- « docker start » redémarre un conteneur arrêté
- Il redémarre avec les mêmes paramètres
- On peut en profiter pour l'attacher avec l'option -a

```
[root@localhost ~]# docker start 617dbe251a64  
617dbe251a64
```

```
[root@localhost ~]# docker ps  
CONTAINER ID        IMAGE       COMMAND          CREATED             STATUS              PORTS      NAMES  
617dbe251a64        almalinux   "ping 127.0.0.1"   8 minutes ago   Up 4 seconds          hopeful_austin
```



# DOCKER CLI

Docker CLI

Docker CLI - TP

## Inspecter un conteneur

- « docker inspect » affiche tous les détails du conteneur
- Le résultat est présenté sous la forme d'un tableau JSON

```
[root@localhost ~]# docker inspect 617dbe251a64
[
  {
    "Id": "617dbe251a6480d53445ca859adb509ead849f2357cba3a5e8ffff65253b4b909",
    "Created": "2022-11-14T15:53:12.607502579Z",
    "Path": "ping",
    "Args": [
      "127.0.0.1"
    ],
    "State": {
      "Status": "running",
      "Running": true,
      "Paused": false,
      "Restarting": false,
      "OOMKilled": false,
      "Dead": false,
      "Pid": 24464,
      "ExitCode": 0,
      "Error": "",
      "StartedAt": "2022-11-14T16:01:37.666701646Z",
      "FinishedAt": "2022-11-14T15:57:22.810446043Z"
    },
    "Image": "sha256:5d0da3dc976460b72c77d94c8a1ad043
  [...]
```



# DOCKER CLI

Docker CLI  
Docker CLI - TP

## Filtrer docker inspect

- `--format='{{.champ1.champ2}}'`
- `--format='{{json .champ2}}'`

```
[root@localhost ~]# docker inspect --format='{{.State.Status}}' 617dbe251a64
Running

[root@localhost ~]# docker inspect --format='{{.NetworkSettings.IPAddress}}' 617dbe251a64
172.17.0.2

[root@localhost ~]# docker inspect --format='{{.State}}' 617dbe251a64
{running true false false false false 24464 0 2022-11-14T16:01:37.666701646Z 2022-11-14T15:57:22.810446043Z <nil>}
```



# DOCKER CLI

Docker CLI

Docker CLI - TP

## Supprimer un conteneur

- « docker rm »
- On ne peut supprimer qu'un conteneur arrêté...
- ... Sauf avec l'option -f

```
[root@localhost ~]# docker rm 617dbe251a64
Error response from daemon: You cannot remove a running container 617dbe251a6480d53445ca859adb509ead849f2357cba3a5e8fff65253b4b909. Stop the container before attempting removal or force remove
[root@localhost ~]# docker rm -f 617dbe251a64
617dbe251a64
```



# LES COMMANDES PRINCIPALES

Docker CLI

[Docker CLI - TP](#)



# DOCKER CLI - TP

## Installation

- Mettez à jour la distribution
- Suivez la procédure d'installation de Docker
- Installez Docker CE
- Démarrez le démon Docker
- Affichez la version de Docker

Docker CLI  
Docker CLI - TP



# DOCKER CLI - TP

Docker CLI  
Docker CLI - TP

## Utilisation de Docker Hub

- Téléchargez la dernière image **ubuntu** à partir du Hub
- Listez les images présentes localement



# DOCKER CLI - TP

Docker CLI  
Docker CLI - TP

## Exécuter un conteneur

- Exécutez un conteneur à partir de l'image **ubuntu**
- Exécutez la commande **ps -ef**
  - à l'intérieur du conteneur
  - depuis l'extérieur
- Listez les conteneurs en exécution
- Listez tous les conteneurs



# DOCKER CLI - TP

Docker CLI  
Docker CLI - TP

## Accéder au terminal d'un conteneur

- Créez un conteneur à l'aide de l'image **ubuntu** et connectez-vous au terminal
- Créez un fichier dans le conteneur puis sortez du conteneur
- Relancez la commande run écrite au premier point de cette diapo  
Que s'est il passé ?



# DOCKER CLI - TP

Docker CLI  
Docker CLI - TP

## Exécution en mode détaché

- Lancer un conteneur basé sur l'image `rockylinux/rockylinux` en exécutant la commande `ping 127.0.0.1 -c 60` en mode détaché
- Listez les conteneurs
- Attendez quelques secondes et listez à nouveau les conteneurs  
Que s'est il passé ?



# DOCKER CLI - TP

Docker CLI  
Docker CLI - TP

## S'attacher et se détacher d'un conteneur

- Lancez un conteneur basé sur l'image **ubuntu** avec la commande **bash** qui tourne à l'intérieur
- Attachez-vous à ce conteneur
- Appuyez sur CTRL + P + Q : que se passe-t-il ?



# DOCKER CLI - TP

Docker CLI  
Docker CLI - TP

## La commande exec

- Exécutez un conteneur en arrière plan en lançant **bash**
- Listez les conteneurs
- Attachez-vous puis détachez-vous du conteneur
- Exécutez à nouveau **bash** à l'intérieur du même conteneur
- Listez les processus à l'intérieur du conteneur.  
Que constatez-vous ?  
Observez les PPID : que constatez-vous ?



# DOCKER CLI - TP

Docker CLI  
Docker CLI - TP

## La commande logs

- Exécutez un conteneur en arrière-plan
- Récupérez l'id du conteneur et consultez les logs
- Consultez les logs en ajoutant l'option -f
- Consultez les logs en ajoutant l'option -f et l'option --tail 10



# DOCKER CLI - TP

Docker CLI  
Docker CLI - TP

## Les commandes stop et start

- Exécutez un conteneur en arrière-plan
- Arrêtez le conteneur
- Redémarrez le conteneur



# DOCKER CLI - TP

Docker CLI  
Docker CLI - TP

## La commande inspect

- Testez la commande inspect, lisez attentivement les différentes informations que vous obtenez
- Testez l'option --format pour sélectionner uniquement l'adresse IP du conteneur
- Utilisez l'option --format pour sélectionner toutes les informations relatives au réseau du conteneur



# DOCKER CLI - TP

Docker CLI  
Docker CLI - TP

## Supprimer un conteneur

- Listez les conteneurs arrêtés à l'aide de l'option --filter
- Supprimez un conteneur arrêté
- Supprimez tous les conteneurs arrêtés



# LES COMMANDES PRINCIPALES

Dans ce chapitre nous avons :

- Découvert la CLI de Docker.
- Découvert comment manipuler la CLI de Docker.



# CHAPITRE 4

# LES IMAGES



# LES IMAGES

- **Les images Docker** nous expliquera comment manipuler les images Docker.
- **Images - TP** nous permettra de manipuler les images Docker.



# LES IMAGES

Les images Docker  
Images - TP



# LES IMAGES DOCKER

## Les couches (layers)

- Une image c'est
  - Une collection de fichiers
  - Des métadonnées
- Elles ont plusieurs couches
- Une couche est une image
- Docker utilise un mécanisme appelé « Copy On Write »
- Les couches sont en lecture seule

Les images Docker  
Images - TP



# LES IMAGES DOCKER

## Les couches (layers)

- Il y a une couche supérieure en écriture pour les conteneurs
- Les couches / images parentes sont en lecture seule
- Les modifications sont appliquées à la couche en écriture
- Lors de la modification d'un fichier depuis une couche en lecture
  - Le mécanisme Copy On Write (COW) copie le fichier sur la couche en écriture
  - Le modifie

Les images Docker  
Images - TP



# LES IMAGES DOCKER

Les images Docker  
Images - TP

## Créer une image

Deux (Trois) méthodes :

- Valider les modifications d'un conteneur en tant qu'image
  - Interactif
  - Utiliser un terminal pour installer les programmes
  - « docker commit »
- Construire à partir d'un Dockerfile
- (Importer une archive dans Docker comme couche de base)



# LES IMAGES DOCKER

Les images Docker  
Images - TP

## A la main

```
[root@localhost ~]# docker run -it almalinux bash
[root@928bb840808c /]# wget
bash: wget: command not found
[root@928bb840808c /]# yum install wget
CentOS Linux 8 - AppStream          3.8 MB/s | 9.6 MB   00:02
CentOS Linux 8 - BaseOS             3.9 MB/s | 8.5 MB   00:02
CentOS Linux 8 - Extras              20 kB/s | 10 kB   00:00
Dependencies resolved.
=====
 Package           Architecture      Version       Repository      Size
=====
Installing:
 wget              x86_64          1.19.5-10.el8    appstream     734 k
Installing dependencies:
 libpsl             x86_64          0.20.2-6.el8     baseos        61 k
 publicsuffix-list-dafsa  noarch      20180723-1.el8    baseos        56 k
Transaction Summary
=====
Install 3 Packages

Total download size: 851 k
Installed size: 2.9 M
Is this ok [y/N]: y

Installed:
  libpsl-0.20.2-6.el8.x86_64      publicsuffix-list-dafsa-20180723-1.el8.noarch      wget-1.19.5-10.el8.x86_64

Complete!
[root@928bb840808c /]# exit
```



# LES IMAGES DOCKER

Les images Docker  
Images - TP

## Lister les différences

- « docker diff » : l'image parente est comparée avec la nouvelle couche
- Liste les fichiers et répertoires qui ont changé

```
[root@localhost ~]# docker diff 928bb840808c
C /root
A /root/.bash_history
C /etc
C /etc/ld.so.cache
A /etc/wgetrc
C /usr
C /usr/lib64
A /usr/lib64/libpsl.so.5
A /usr/lib64/libpsl.so.5.3.1
C /usr/bin
A /usr/bin/wget
```



# LES IMAGES DOCKER

Les images Docker  
Images - TP

## Et le commit !

- Attention ! Le nom du dépôt est basé sur utilisateur/application

```
[root@localhost ~]# docker commit 928bb840808c soli/coursdocker:1.0
sha256:7c66392295c0a57b326eabc53ba78b97f95b78179fd65651ce92f392b3e530ae
```

```
[root@localhost ~]# docker images
REPOSITORY          TAG      IMAGE ID   CREATED        SIZE
soli/coursdocker   1.0      7c66392295c0   5 seconds ago  281MB
ubuntu              latest    ba6accce2d29   4 weeks ago   72.8MB
ubuntu              21.04    de6f83bfe0b6   6 weeks ago   80MB
hello-world         latest    feb5d9fea6a5   7 weeks ago   13.3kB
almalinux           latest    5d0da3dc9764   8 weeks ago   231MB
ubuntu              14.04    13b66b487594   7 months ago  197MB
```



# LES IMAGES DOCKER

## Dockerfile

- Fichier de configuration qui contient les instructions pour la construction d'une image docker
- Plus efficace que le docker commit (on peut le versionner)
- Peut être utilisé dans processus de développement et d'intégration

Les images Docker  
Images - TP



# LES IMAGES DOCKER

## Ecrire son Dockerfile

- On crée un fichier « Dockerfile » dans un dossier
- On écrit les instructions de création de l'image
  - Image de base
  - Programmes à installer
  - Commande à exécuter au lancement
- On exécute la commande « docker build » pour construire l'image à partir du fichier

Les images Docker  
Images - TP



# LES IMAGES DOCKER

Les images Docker  
Images - TP

## Les instructions du Dockerfile

- « **FROM** » spécifie l'image de base à utiliser
  - Première instruction du fichier
  - Peut être spécifiée plusieurs fois pour créer plusieurs images
  - Chaque FROM est le début d'une nouvelle image
- « **RUN** » spécifie une commande à exécuter
  - Utilisée pour installer des packages, des bibliothèques
  - On exécute la commande « docker build » pour construire l'image à partir du fichier



# LES IMAGES DOCKER

Les images Docker  
Images - TP

## Exemple

```
[root@localhost ~]# cat monappli/Dockerfile
# mon appli de test !
FROM almalinux:latest
RUN yum install -y wget

[root@localhost ~]# docker build -t soli/monappli:1.0 monappli
Sending build context to Docker daemon 2.048kB
Step 1/2 : FROM almalinux:latest
--> 5d0da3dc9764
Step 2/2 : RUN yum install -y wget
--> Running in 35947e7f9d3c # Création d'un conteneur intermédiaire temporaire

Installed:
libpsl-0.20.2-6.el8.x86_64      publicsuffix-list-dafsa-20180723-1.el8.noarch
wget-1.19.5-10.el8.x86_64

Complete!
Removing intermediate container 35947e7f9d3c # Une fois que c'est enregistré comme une nouvelle couche, suppression du conteneur
--> 8879ac65821a
Successfully built 8879ac65821a
Successfully tagged soli/monappli:1.0

[root@localhost ~]# docker images
REPOSITORY          TAG        IMAGE ID       CREATED        SIZE
soli/monappli      1.0        8879ac65821a   4 seconds ago  281MB
almalinux           latest     5d0da3dc9764   8 weeks ago   231MB
```



# LES IMAGES DOCKER

Les images Docker  
Images - TP

## Contexte de build

- Le contexte c'est le contenu du répertoire où se trouve le Dockerfile
- Il est envoyé au docker daemon sous forme d'une archive
- Docker daemon va construire l'image en utilisant les fichiers disponibles dans le contexte

```
[root@localhost ~]# docker build -t soli/monappli:1.0 monappli
Sending build context to Docker daemon 2.048kB
```



# LES IMAGES DOCKER

Les images Docker  
Images - TP

## Cache de build

- Docker enregistre un instantané de l'image après chaque étape de construction
- Avant d'exécuter une étape, docker vérifie s'il ne l'a pas déjà
  - Si oui il utilise le résultat précédent récupéré dans le cache
  - On peut désactiver ce comportement avec --no-cache
- Le moindre changement d'instruction dans le Dockerfile annule le cache

```
[root@localhost ~]# docker build -t soli/monappli:1.0 monappli
Sending build context to Docker daemon 2.048kB
Step 1/2 : FROM almalinux:latest
 ---> 5d0da3dc9764
Step 2/2 : RUN yum install -y wget
 ---> Using cache # <-
 ---> 8879ac65821a
Successfully built 8879ac65821a
Successfully tagged soli/monappli:1.0
```



# LES IMAGES DOCKER

Les images Docker  
Images - TP

## Lister les couches

- « docker history » liste les couches utilisées pour créer une image
- Affiche la date de création, la taille et la commande qui a été exécutée

[root@localhost ~]# docker history soli/monappli:1.0				
IMAGE	CREATED	CREATED BY	SIZE	COMMENT
8879ac65821a	15 minutes ago	/bin/sh -c yum install -y wget	49.7MB	
5d0da3dc9764	8 weeks ago	/bin/sh -c #(nop) CMD ["/bin/bash"]	0B	
<missing>	8 weeks ago	/bin/sh -c #(nop) LABEL org.label-schema.sc...	0B	
<missing>	8 weeks ago	/bin/sh -c #(nop) ADD file:805cb5e15fb6e0bb0...	231MB	



# LES IMAGES DOCKER

Les images Docker  
Images - TP

## Dockerfile : instruction CMD

- « CMD » spécifie la commande par défaut exécutée lorsque le conteneur est créé
- Spécifié une seule fois
- Ecrasé par les arguments
- Sinon c'est la dernière qui est prise en compte
- Format :
  - Shell : CMD ping 127.0.0.1 -c 30
  - EXEC : CMD ["ping","127.0.0.1","-c","30"]



# LES IMAGES DOCKER

Les images Docker  
Images - TP

## Dockerfile : instruction ENTRYPOINT

- Ressemble beaucoup à CMD
- N'est pas écrasé par les arguments, il est complété par les arguments qu'on passe au lancement OU par ce qui est contenu dans CMD
- Il peut-être écrasé avec l'option --entrypoint
- Tout ça n'est valable que lorsque l'instruction est écrite sous forme de tableau !



# LES IMAGES DOCKER

Les images Docker  
Images - TP

## Dockerfile : instruction COPY

- « COPY »
- Copie les fichiers à partir de la source (hôte) vers la destination dans le système de fichier du conteneur
- Le chemin de la source doit être dans le contexte de build
- Si la source est un répertoire, tous les fichiers du répertoire sont copiés, pas le répertoire lui-même



# LES IMAGES DOCKER

Les images Docker  
Images - TP

## Dockerfile : instruction ADD

- « ADD »
- Copie les fichiers à partir de la source (hôte) vers la destination dans le système de fichier du conteneur
- Permet d'extraire les fichiers d'une archive
- Permet d'utiliser une URL



# LES IMAGES DOCKER

## Dockerfile : instruction LABEL

- « LABEL variable=valeur »
- Ajoute des metadatas sur l'image
- Exemple :

```
LABEL maintainer="John Doe jdoe@gmail.com"
```



# LES IMAGES DOCKER

## Dockerfile : bonnes pratiques

- Faire les étapes les plus générales et les plus longues en premier (utilisation du cache !)
- Si les commandes sont longues, leur attribuer leur propre couche
- Utiliser && et \ pour combiner les commandes similaires

Les images Docker  
Images - TP



# LES IMAGES

Les images Docker  
**Images - TP**



## IMAGES - TP

Les images Docker  
Images - TP

### Modifications dans un conteneur

- Exécutez un conteneur à partir de l'image `rockylinux/rockylinux` et installez `wget` à l'intérieur
- Comparez le conteneur modifié et l'image de base



## IMAGES - TP

Les images Docker  
Images - TP

### Création d'une nouvelle image

- Créez une nouvelle image à partir du conteneur modifié
- Créez un conteneur à partir de la nouvelle image
- Vérifiez que vous avez bien le logiciel `wget` installé



# IMAGES - TP

Les images Docker  
Images - TP

## Dockerfile

- Créez un fichier Dockerfile dans un nouveau répertoire
- Remplissez le Dockerfile pour qu'il utilise **rockylinux/rockylinux** comme image de base, qu'il y ait une mise à jour et que wget soit installé
- Construisez la nouvelle image
- Listez les images
- Modifiez le Dockerfile pour ajouter l'installation d'un nouveau paquet, zip
- Construisez la nouvelle image  
Observez l'utilisation du cache
- Visualisez l'historique de la construction des images
- Modifiez le fichier Dockerfile pour que les deux installations soient faites avec la même commande  
Que remarquez-vous lors de la construction ?



# LES IMAGES

Dans ce chapitre nous avons :

- Vu ce que sont les images Docker.
- Vu comment manipuler les images Docker.



# CHAPITRE 5

# LE RESEAU



# LE RESEAU

- **Le réseau avec Docker** nous expliquera comment on gère le réseau avec Docker.
- **Réseau - TP** nous permet de tester le réseau avec Docker.



# LE RESEAU

Le réseau avec Docker  
Réseau - TP



# LE RÉSEAU AVEC DOCKER

Le réseau avec Docker  
Réseau - TP

## Modèle réseau

- Les conteneurs n'ont pas d'adresse IPV4 publique
- Ils ont une adresse privée
- Les services doivent être exposés port par port
- Les ports de conteneurs doivent être mappés sur les ports de l'hôte pour éviter les conflits
- Au démarrage, Docker crée une interface virtuelle docker0 sur l'hôte avec une adresse privée aléatoire



# LE RÉSEAU AVEC DOCKER

Le réseau avec Docker  
Réseau - TP

## Bridge

- L'interface docker0 est un pont ethernet virtuel
- Docker0 commute les paquets ethernet entre deux interfaces comme un pont ou un commutateur physique
  - De l'hôte vers un conteneur
  - D'un conteneur vers un autre conteneur
- Chaque nouveau conteneur obtient une interface attachée au pont



# LE RÉSEAU AVEC DOCKER

Le réseau avec Docker  
Réseau - TP

## docker0

```
[root@localhost ~]# ip a
1: lo: <LOOPBACK,UP,LOWER_UP> mtu 65536 qdisc noqueue state UNKNOWN group default qlen 1000
    link/loopback 00:00:00:00:00:00 brd 00:00:00:00:00:00
    inet 127.0.0.1/8 scope host lo
        valid_lft forever preferred_lft forever
    inet6 ::1/128 scope host
        valid_lft forever preferred_lft forever
[...]
5: docker0: <NO-CARRIER,BROADCAST,MULTICAST,UP> mtu 1500 qdisc noqueue state DOWN group default
    link/ether 02:42:a3:52:fb:f5 brd ff:ff:ff:ff:ff:ff
    inet 172.17.0.1/16 brd 172.17.255.255 scope global docker0
        valid_lft forever preferred_lft forever
    inet6 fe80::42:a3ff:fe52:fbf5/64 scope link
        valid_lft forever preferred_lft forever
```

```
[root@localhost ~]# bridge link show
4: virbr0-nic: <BROADCAST,MULTICAST> mtu 1500 master virbr0 state disabled priority 32 cost 100
```

```
[root@localhost ~]# docker run -d -it ubuntu
04737751d692668b9cf914ba7f3a9ecc1ab7b4868819e241fbfc11b83a01cbd0
```

```
[root@localhost ~]# bridge link show
4: virbr0-nic: <BROADCAST,MULTICAST> mtu 1500 master virbr0 state disabled priority 32 cost 100
37: veth66f55c6@if36: <BROADCAST,MULTICAST,UP,LOWER_UP> mtu 1500 master docker0 state forwarding priority 32 cost 2
```



# LE RÉSEAU AVEC DOCKER

Le réseau avec Docker  
Réseau - TP

## Réseaux par défaut

- Docker initialise automatiquement 3 réseaux
- Le réseau bridge est le docker0
- Par défaut tous les conteneurs sont connectés au réseau bridge
- Si on connecte le conteneur au réseau « none » il n'aura pas d'interface réseau
- Si on connecte le conteneur au réseau host, le conteneur sera sur la même pile réseau que son hôte

```
[root@localhost ~]# docker network ls
NETWORK ID      NAME      DRIVER      SCOPE
d3d6f4c3cf4a    bridge    bridge      local
176c6834a14e    host      host       local
e85f0769c39c    none      null       local
```



# LE RÉSEAU AVEC DOCKER

Le réseau avec Docker  
Réseau - TP

## Inspection du réseau

```
[root@localhost ~]# docker network inspect bridge
[{"Name": "bridge", "Id": "d3d6f4c3cf4ad55b340c1157ac5bbc2dea0eeb302f1efb869c6ee68103d34490", "Created": "2022-11-14T09:33:29.372571434-05:00", "Scope": "local", "Driver": "bridge", "EnableIPv6": false, "IPAM": {"Driver": "default", "Options": null, "Config": [{"Subnet": "172.17.0.0/16", "Gateway": "172.17.0.1"}]}, "Containers": {"04737751d692668b9cf914ba7f3a9ecc1ab7b4868819e241fbfc11b83a01cbd0": {"Name": "gracious_rubin", "EndpointID": "12f569b27d841f110d385709b835286547eef6a1802526c343896a86cd752dcc", "MacAddress": "02:42:ac:11:00:02", "IPv4Address": "172.17.0.2/16", "IPv6Address": ""}}, "Options": {"com.docker.network.bridge.default_bridge": "true", "com.docker.network.bridge.enable_icc": "true", "com.docker.network.bridge.enable_ip_masquerade": "true", "com.docker.network.bridge.host_binding_ipv4": "0.0.0.0", "com.docker.network.bridge.mtu": "1500"}]}
```



# LE RÉSEAU AVEC DOCKER

Le réseau avec Docker  
Réseau - TP

## Créer un réseau

Deux types de réseaux :

- Bridge, similaire au bridge docker0
- Overlay, un bridge à travers plusieurs hôtes

```
docker network create <network name>
```



# LE RÉSEAU AVEC DOCKER

Le réseau avec Docker  
Réseau - TP

## DNS intégré

- Le DNS intégré permet la découverte des containers créés avec un nom valide ou un alias réseau
- Intégré au daemon docker
- Permet aux conteneurs de communiquer sur le même bridge

```
[root@localhost ~]# docker run -d -it --net=mon_bridge --name=host1 ubuntu  
07f1d021f3a80d08de768984f28e746b12517592b5ef1e5b264dd8242a004c6f  
[root@localhost ~]# docker run -it --net=mon_bridge --name=host2 ubuntu  
root@00facc8e4f27:/# ping host1  
PING host1 (172.18.0.3) 56(84) bytes of data.  
64 bytes from host1.mon_bridge (172.18.0.3): icmp_seq=1 ttl=64 time=0.065 ms  
64 bytes from host1.mon_bridge (172.18.0.3): icmp_seq=2 ttl=64 time=0.081 ms
```



# LE RÉSEAU AVEC DOCKER

Le réseau avec Docker  
Réseau - TP

## Réseaux multiples

- Les conteneurs peuvent être connectés à plusieurs réseaux

```
docker network connect <network> <container>
```



# LE RÉSEAU AVEC DOCKER

Le réseau avec Docker  
Réseau - TP

## Mapping des ports

- Les conteneurs en cours d'exécution dans un réseau bridge ne sont accessibles que par l'hôte sur lequel le bridge est monté
- Pour qu'un conteneur soit accessible à l'extérieur, il faut exposer les ports du conteneur et les mapper sur des ports de l'hôte
- Le conteneur sera accessible via le port mappé de l'hôte
- Les ports peuvent être mappés manuellement ou automatiquement



# LE RÉSEAU AVEC DOCKER

Le réseau avec Docker  
Réseau - TP

## Mapping manuel

- « docker run -p [host port]:[container port] »
- On peut utiliser plusieurs fois l'option -p pour mapper plusieurs ports
- Pour visualiser le mapping
  - « docker port »

```
[root@localhost ~]# docker run -d -p 8080:80 nginx
Status: Downloaded newer image for nginx:latest
B2ab67bca024e2bc8d70c892876f8fc6ed7fd8a154ab2fe8e752f7a4b8ed4dbc

[root@localhost ~]# docker ps
CONTAINER ID        IMAGE       COMMAND                  CREATED             STATUS              PORTS                 NAMES
b2ab67bca024        nginx      "/docker-entrypoint..."   About a minute ago   Up About a minute   0.0.0.0:8080->80/tcp, :::8080->80/tcp   elated_austin
04737751d692        ubuntu     "bash"                  19 minutes ago      Up 19 minutes

[root@localhost ~]# docker port b2ab67bca024
80/tcp -> 0.0.0.0:8080
80/tcp -> :::8080
```



# LE RÉSEAU AVEC DOCKER

Le réseau avec Docker  
Réseau - TP

## Mapping automatique

- « docker run -P »
- Mappe automatiquement les ports exposés dans le conteneur sur un numéro de port de l'hôte
- Les numéros de port utilisés par l'hôte vont de 32768 à 65535
- Utiliser l'instruction EXPOSE permet d'exposer les ports avec un Dockerfile

```
[root@localhost ~]# docker run -d -P nginx
d3c41fefb237b8fc6e21a4cfdea40e4c28c7e7b9e5fe9d0e09d621fe361781f7

[root@localhost ~]# docker ps
CONTAINER ID   IMAGE      COMMAND       CREATED          STATUS          PORTS          NAMES
d3c41fefb237   nginx      "/docker-entrypoint..."   17 seconds ago   Up 16 seconds   0.0.0.0:49153->80/tcp, :::49153->80/tcp   elastic_kepler

[root@localhost ~]# docker port d3c41fefb237
80/tcp -> 0.0.0.0:49153
80/tcp -> :::49153
```



# LE RESEAU

Le réseau avec Docker  
**Réseau - TP**



# RÉSEAU - TP

Le réseau avec Docker  
Réseau - TP

## Réseau par défaut

- Lancez un conteneur en arrière-plan
- Utilisez la commande suivante pour regarder le réseau bridge

```
[root@rocky ~]# docker network inspect bridge
```

- Lancez un autre conteneur et testez le réseau à l'aide de la commande ping et des adresses IP des conteneurs



# RÉSEAU - TP

Le réseau avec Docker  
Réseau - TP

## Création d'un bridge alternatif

- Créez un bridge
- Lancez plusieurs conteneurs nommés puis testez le réseau sur le nouveau bridge



# RÉSEAU - TP

Le réseau avec Docker  
Réseau - TP

## Connexion à plusieurs réseaux

- Lancez un nouveau conteneur nommé en mode interactif et essayez de « pinguer » le précédent
- Connectez-vous au même réseau, retestez le ping



# RÉSEAU - TP

Le réseau avec Docker  
Réseau - TP

## Le mapping de ports

- Lancez un conteneur basé sur l'image **nginx** avec un mapping manuel
- Listez les conteneurs
- Regardez les infos sur les ports avec la commande adaptée
- Lancez un conteneur basé sur l'image **nginx** avec un mapping automatique
- Sur quel port le port 80 de nginx est-il mappé ?
- Créez une nouvelle image qui expose directement le port 80 automatiquement.



# RÉSEAU - TP

Le réseau avec Docker  
Réseau - TP

## Tests avec haproxy

Nous allons créer une application avec 3 conteneurs dans un réseau privé :

- web1 : un serveur nginx non mappé qui doit afficher Server ONE
- web2 : un serveur nginx non mappé qui doit afficher Server TWO
- myhaproxy : un serveur haproxy mappé, qui recevra en FrontEnd les requêtes HTTP des clients sur le port 80 du Docker Host et qui redirigera en BackEnd vers les 2 serveurs web1 et web2



# RÉSEAU - TP

Le réseau avec Docker  
Réseau - TP

- Créez un réseau de type bridge nommé « web »
- Vérifiez qu'il apparaît bien dans la liste des réseaux
- Créez 2 images **nginx1** et **nginx2** avec des fichiers index.html spécifiques

Par exemple en utilisant le fichier suivant et en l'adaptant :

```
[almalinux@host11 ~]$ cat web1/Dockerfile
FROM nginx
COPY index.html /usr/share/nginx/html
```

```
[almalinux@host11 ~]$ cat web1/index.html
Server ONE
```



# RÉSEAU - TP

Le réseau avec Docker  
Réseau - TP

- Créez l'image myhaproxy avec le fichier de configuration fourni :

```
[almalinux@host11 ~]$ cat myhaproxy/Dockerfile
FROM haproxy
COPY haproxy.cfg /usr/local/etc/haproxy/haproxy.cfg
```

```
[almalinux@host11 ~]$ cat myhaproxy/haproxy.cfg
frontend http_frontend
bind *:80
default_backend http_backend
backend http_backend
mode http
balance roundrobin
server server1 web1:80 check
server server2 web2:80 check
```



# RÉSEAU - TP

Le réseau avec Docker  
Réseau - TP

- Lancez les 3 conteneurs avec les noms adaptés quand c'est nécessaire (voir fichier au dessus)
- Testez la commande curl 0 plusieurs fois de suite, expliquez ce que vous observez.



# LE RESEAU

Dans ce chapitre nous avons :

- Vu comment manipuler les réseaux avec Docker.
- Manipulé les réseaux avec Docker.



# CHAPITRE 6

# **LES VOLUMES**



# LES VOLUMES

- **Les volumes avec Docker** nous expliquera comment on gère les données avec Docker.
- **Volumes - TP** nous permettra de manipuler les volumes Docker.



# LES VOLUMES

Les volumes avec Docker  
Volumes - TP



# LES VOLUMES AVEC DOCKER

## Qu'est ce que c'est ?

- Répertoire dans un conteneur
- Dédié à la persistance des données indépendamment du cycle de vie du conteneur
  - Persistant si conteneur supprimé
  - Peut être mappé sur un répertoire sur le hôte
  - Pas affecté par le COW
  - Si on crée une image à partir d'un conteneur, le contenu des volumes n'est pas intégré à l'image

Les volumes avec Docker  
Volumes - TP



# LES VOLUMES AVEC DOCKER

## Pourquoi les volumes ?

- Partager les données entre conteneur
- Contourner le COW pour utiliser les performances disques de l'hôte
- Partager un répertoire hôte avec un conteneur
- Partager un fichier unique entre l'hôte et le conteneur

Les volumes avec Docker  
Volumes - TP



# LES VOLUMES AVEC DOCKER

Les volumes avec Docker  
Volumes - TP

## Créer un volume

- Créer : docker volume create [--name nom\_volume]
- Lister : docker volume ls

```
[root@localhost ~]# docker volume create --name monvolume
Monvolume

[root@localhost ~]# docker volume ls
DRIVER      VOLUME NAME
local      3e2e39d38b2956061966f3b32becb3d95fe6abf8e0a57a6d60981c90272daba9
local      185bae846567a4d0cb174aabca37d2a86297d62b0f35e09853042f368438c038
local      fd4a554f9f65330005920ce0df5fcfebc971a7f76162fb5223822692743740a4
local      monvolume
```



# LES VOLUMES AVEC DOCKER

Les volumes avec Docker  
Volumes - TP

## Montage des volumes

```
[root@localhost ~]# docker run -it -v monvolume:/www/monvolume almalinux bash
[root@702e540d4965 /]# touch /www/monvolume/fichiertest
[root@702e540d4965 /]# exit
[root@localhost ~]# ls -l /var/lib/docker/volumes/monvolume/_data/
total 0
-rw-r--r--. 1 root root 0 14 nov. 14:09 fichiertest
```



# LES VOLUMES AVEC DOCKER

Les volumes avec Docker  
Volumes - TP

## Inspection des volumes

```
[root@localhost ~]# docker volume inspect monvolume
[
  {
    "CreatedAt": "2022-11-14T14:09:13-05:00",
    "Driver": "local",
    "Labels": {},
    "Mountpoint": "/var/lib/docker/volumes/monvolume/_data",
    "Name": "monvolume",
    "Options": {},
    "Scope": "local"
  }
]
```



# LES VOLUMES AVEC DOCKER

Les volumes avec Docker  
Volumes - TP

## Suppression des volumes

- Non supprimés lorsque vous supprimez un conteneur
  - « docker volume rm »
- Possible de supprimer les volumes associés à un conteneur lorsque l'on supprime celui-ci
  - « docker rm -v »
- On ne peut pas supprimer un volume utilisé par conteneur, même arrêté

```
[root@localhost ~]# docker volume rm monvolume
Error response from daemon: remove monvolume: volume is in use - [702e540d49654f3ca745f4b78d257002ed32511613efc890bd57da5798874fb1]
[root@localhost ~]# docker volume create --name monvolume2
monvolume2
[root@localhost ~]# docker volume rm monvolume2
monvolume2
```



# LES VOLUMES AVEC DOCKER

Les volumes avec Docker  
Volumes - TP

## Volumes d'hôtes

- Lors de l'exécution d'un conteneur, vous pouvez mapper des dossiers de l'hôte sur un volume
- Les fichiers du dossier hôte seront présents dans le volume
- Les modifications d'un côté affectent l'autre
- S'il n'existe pas, le chemin sera créé
- Si le dossier existe, les fichiers sont remplacés par ceux de l'hôte



# LES VOLUMES AVEC DOCKER

## Volumes partagés

- Un volume peut être monté dans plusieurs conteneurs
- Permet de partager les données entre conteneurs
- Exemple :
  - Un conteneur écrit des données dans le volume
  - Un autre exécute une application qui lit les données et génère des graphes
- Attention aux conflits potentiels si plusieurs conteneurs ont accès en écriture au même volume !

Les volumes avec Docker  
Volumes - TP



# LES VOLUMES AVEC DOCKER

## Volumes dans un Dockerfile

- L'instruction `VOLUME` crée un point de montage
- Ce n'est pas ici qu'on fait le mapping avec l'hôte
- Les volumes sont initialisés lorsque le conteneur est exécuté
- Syntaxe
  - `VOLUME /vol1`
  - `VOLUME /www/site1 /www/site2`
  - `VOLUME ["vol1","vol2"]`

Les volumes avec Docker  
Volumes - TP



# LES VOLUMES

Les volumes avec Docker  
**Volumes - TP**



# VOLUMES - TP

Les volumes avec Docker  
**Volumes - TP**

## Création des volumes

- Créez un volume **vol1**
- Listez les volumes
- Montez le **vol1** dans un conteneur ubuntu
- Créez un fichier dans un conteneur ubuntu
- Quittez le conteneur avec **exit**
- Créez une image à partir du conteneurs
- Lancez un conteneur à partir de la nouvelle image.  
Que remarquez-vous ?



# VOLUMES - TP

Les volumes avec Docker  
**Volumes - TP**

## Utilisation des volumes

- Listez les volumes
- Affichez les caractéristiques d'un volume
- Vérifiez la persistance des données dans le volume
- Lancez un nouveau conteneur et accédez au même volume, créez un autre fichier
- Passez le conteneur en arrière plan puis connectez vous au premier en utilisant la commande **exec**  
Qu'observez-vous ?



## VOLUMES - TP

Les volumes avec Docker  
**Volumes - TP**

# Suppression des volumes

- Listez les volumes
- Supprimez le volume vol1



# VOLUMES - TP

Les volumes avec Docker  
**Volumes - TP**

## Montez un volume host

- Montez un volume host, par exemple un dossier que vous aurez créé et dans lequel vous aurez placé un fichier
- Montez un volume pour les logs d'un conteneur nginx
- Accédez au serveur à l'aide de la commande **curl**
- Inspectez le volume des logs
- Regardez dans le dossier sur l'hôte



## VOLUMES - TP

Les volumes avec Docker  
**Volumes - TP**

### Les volumes dans un Dockerfile

- Testez l'instruction **VOLUME** dans un Dockerfile.
- Lancez un conteneur à partir de l'image ainsi construite
- Trouvez les répertoires correspondants aux volumes sur l'hôte



## VOLUMES - TP

Les volumes avec Docker  
**Volumes - TP**

## HAPROXY avec les volumes

- Reprendre l'exercice **HAPROXY**, mais lancez les conteneurs avec les images standards, et pour modifier les fichiers index.html et haproxy.cfg, on passe par le montage de volumes des répertoires et fichiers créés précédemment.
- Lancez les serveurs web1 et web2
- Lancez haproxy



# LES VOLUMES

Dans ce chapitre nous avons :

- Vu comment utiliser les volumes pour manipuler les données avec Docker.
- Manipulé les volumes pour gérer les données avec Docker.



## CHAPITRE 7

# DOCKER COMPOSE



# DOCKER COMPOSE

- **Docker compose** dans lequel nous réaliserons une application répartie dans plusieurs conteneurs.
- **Docker compose - TP** nous permettra de manipuler l'outil docker compose.



# DOCKER COMPOSE

Docker compose

Docker compose - TP



# DOCKER COMPOSE

## Qu'est ce que c'est ?

- Docker Compose est un outil pour créer et gérer des applications multiconteneurs
- Les conteneurs sont tous définis dans un seul fichier appelé docker-compose.yml
- Les liens entre conteneurs sont définis
- Compose lancera tous vos conteneurs en une seule commande

Docker compose  
Docker compose - TP



# DOCKER COMPOSE

Docker compose  
Docker compose - TP

## Comment ça fonctionne ?

- Chaque conteneur gère un composant / service particulier de votre application
  - Web front end
  - User authentication
  - Payments
  - Database



# DOCKER COMPOSE

Docker compose  
Docker compose - TP

## Le fichier docker-compose.yml

- Le fichier docker-compose.yml définit les services qui composent une application
- Chaque « service » contient les instructions pour construire et gérer un conteneur

```
services:  
  web:  
    build: .  
    ports:  
      - "5000:5000"  
    volumes:  
      - .:/code  
  redis:  
    image: redis
```



# DOCKER COMPOSE

## Instruction build

- Build définit le chemin d'accès au Dockerfile qui sera utilisé pour créer l'image
- Le conteneur sera utilisé à l'aide de l'image construite
- Le chemin d'accès à la construction peut être relatif par rapport à l'emplacement du fichier yml

```
services:  
  web:  
    build: . # <--- ici  
    ports:  
      - "5000:5000"  
    volumes:  
      - .:/code  
  redis:  
    image: redis
```

Docker compose  
Docker compose - TP



# DOCKER COMPOSE

Docker compose  
Docker compose - TP

## Instruction image

- Définit l'image qui sera utilisée pour exécuter le conteneur
- L'image peut être locale ou distante
- On peut spécifier un tag ou un ID d'image
- Tous les services doivent comporter une instruction build ou image

```
services:  
  web:  
    build: .  
    ports:  
      - "5000:5000"  
    volumes:  
      - .:/code  
  redis:  
    image: redis # <--- ici
```



# DOCKER COMPOSE

## Le réseau

- Le fichier docker-compose permet de gérer les réseaux
- On doit déclarer les réseaux qu'il doit utiliser
- On peut spécifier pour chaque service à quels réseaux il appartient

Docker compose  
Docker compose - TP



# DOCKER COMPOSE

## Le réseau - Exemple

```
services:  
  web:  
    build: .  
    ports:  
      - "5000:5000"  
    volumes:  
      - .:/code  
    networks: # <--- l'utilisation  
      - web  
  redis:  
    image: redis  
    networks: # <--- l'utilisation  
      - web  
networks: # <--- la déclaration  
  web:
```

Docker compose  
Docker compose - TP



# DOCKER COMPOSE

## Exécuter l'application

- docker-compose up
- La commande Up sert à :
  - Créer l'image pour chaque service
  - Créer et démarrer les conteneurs
- Les conteneurs peuvent tous fonctionner au premier plan ou en mode détaché

Docker compose  
Docker compose - TP



# DOCKER COMPOSE

## Exécuter l'application

Docker compose  
Docker compose - TP

```
docker@node1:~/HelloRedis-master$ docker-compose up
Pulling redis (redis:latest)...
latest: Pulling from library/redis
065132d9f705: Pull complete
Status: Downloaded newer image for redis:latest
Building javaclient
Step 1/6 : FROM java:7
7: Pulling from library/java
6263baad4f89: Pull complete
Creating helloredismaster_redis_1 ...
Creating helloredismaster_javaclient_1 ...
Attaching to helloredismaster_redis_1, helloredismaster_javaclient_1
redis_1 | 1:C 05 Oct 04:39:55.519 # 0000o000o000o Redis is starting o000o000o000o
redis_1 | 1:M 05 Oct 04:39:55.521 * Ready to accept connections
javaclient_1 | Server is running: PONG
javaclient_1 | books_count = null
```



# DOCKER COMPOSE

Docker compose  
Docker compose - TP

## Visualiser les conteneurs

- docker-compose ps permet de lister les services lancés par Compose
- Cela affichera uniquement les services qui ont été lancés depuis Compose tel que définis dans le fichier **docker-compose.yml**
- La commande doit être exécutée dans le dossier avec le fichier yml

```
docker@node1:~/HelloRedis-master$ docker-compose ps
Name                  Command           State Ports
-----                -----
helloredismaster_javaclient_1  java HelloRedis      Up
helloredismaster_redis_1       docker-entrypoint.sh redis ... Up      6379/tcp
```



# DOCKER COMPOSE

## start et stop

- Arrêt d'un service
  - docker-compose stop <service name>
- Arrêt de tous les services
  - docker-compose stop
- Redémarrage d'un service
  - docker-compose start <service name>
- Redémarrage de tous les services
  - docker-compose start

Docker compose  
Docker compose - TP



# DOCKER COMPOSE

## Suppression

- Vous pouvez supprimer manuellement chaque conteneur de service avec la commande `docker rm`
- Ou `docker-compose rm` pour supprimer tous les conteneurs de service qui ont été arrêtés
- Spécifiez un service à supprimer `docker-compose rm <service name>`
- Utilisez l'option `-v` pour supprimer les volumes associés `docker-compose rm -v <service name>`

Docker compose  
Docker compose - TP



# DOCKER COMPOSE

## Logs

```
docker-compose logs [<service name>]
```

- Si un service n'est pas spécifié, le journal agrégé de tous les conteneurs sera affiché



# DOCKER COMPOSE

Docker compose  
Docker compose - TP

## Scale

- Dans une architecture de micro-service, nous avons la flexibilité d'étendre un service particulier pour gérer une charge plus grande

```
docker-compose up -d --scale <service name>=<instances>
```

```
[root@localhost HelloRedis]# docker-compose up -d --scale javaclient=2
[+] Running 3/3
  □ Container helloredis-redis-1    Running          0.0s
  □ Container helloredis-javaclient-2 Started        1.7s
  □ Container helloredis-javaclient-1 Started        1.5s
```



# DOCKER COMPOSE

## Variables

- Les variables d'environnement peuvent être utilisées dans votre fichier de configuration yml
- Syntaxe **\$VARIABLE** ou  **\${VARIABLE}**

Docker compose  
Docker compose - TP



# DOCKER COMPOSE

Docker compose  
**Docker compose - TP**



# DOCKER COMPOSE - TP

Docker compose  
Docker compose - TP

## Version de Docker Compose

Attention, avec les nouvelles version de Docker, Docker compose s'intègre directement, ce n'est plus à installer à côté :

- Avant on utilisait un programme à côté de Docker : docker-compose
- Maintenant c'est une commande de Docker : docker compose



# DOCKER COMPOSE - TP

Docker compose  
Docker compose - TP

## Nginx avec Docker Compose

Ecrire un fichier Docker Compose qui permet de lancer un conteneur nginx :

- Avec le port 80 du conteneur mappé sur le port de votre choix de l'hôte
- Avec un volume monté à l'endroit où est l'index.html par défaut de nginx pour personnaliser celui-ci



# DOCKER COMPOSE - TP

Docker compose  
**Docker compose - TP**

## Reprise du TP HAProxy

- Reprenez le TP HAproxy et reformulez-le avec docker-compose
- Lancez les services avec docker-compose



# DOCKER COMPOSE

Dans ce chapitre nous avons :

- Vu comment utiliser compose pour combiner plusieurs conteneurs.
- Manipulé Docker Compose.



# CHAPITRE 8

# L'API



# L'API

- **Docker API** nous présente l'API de Docker.
- **Docker API - TP** nous permet de manipuler l'API.



# L'API

Docker API  
Docker API - TP



# DOCKER API

## API Docker

- Fournie par Docker
- Permet d'interagir avec le daemon docker
- Accessible via un client HTTP (wget ou curl par exemple)
- Accessible via un langage de programmation (bibliothèque http)

Docker API  
Docker API - TP



# DOCKER API

Docker API

Docker API - TP

## REST

- Representational State Transfer
- Architecture qui permet de construire des applications web
- Ensemble de conventions
- Repose sur le protocole HTTP
- Règles à respecter :
  - URI = identifiant de ressource
  - Verbes HTTP = identifiant d'opération



# DOCKER API

## URI

- Uniform Resource Identifier
- Permet d'identifier une ressource
- Une application doit construire ses URI de manière précise
- La hiérarchie des ressources est représentée dans l'URI

Docker API  
Docker API - TP



# URI - Exemples

- Lister des conteneurs
  - <http://localhost/containers/json>
- Filtrer des conteneurs
  - [http://localhost/containers/json?name=mon\\_conteneur&state=running](http://localhost/containers/json?name=mon_conteneur&state=running)
- Afficher les informations d'un conteneur en particulier
  - <http://localhost/containers/3/json>



# DOCKER API

## URI - Hiérarchie

- Le système de hiérarchie permet d'avoir un système clair et compréhensible :
  - `http://localhost/containers/3/json` => les informations d'un conteneur
  - `http://localhost/containers/3/logs` => les logs d'un conteneur
  - `http://localhost/containers/3/top` => les processus dans un conteneur
  - `http://localhost/containers/json` => tous les conteneurs

Docker API

Docker API - TP



# DOCKER API

## HTTP - Les verbes

- Quatre verbes principaux pour le CRUD
- CRUD = Create, Read, Update, Delete
  - Créer (create) => POST
  - Afficher (read) => GET
  - Mettre à jour (update) => PUT
  - Supprimer (delete) => DELETE

Docker API

Docker API - TP



# DOCKER API

Docker API  
Docker API - TP

## HTTP - Les requêtes

- Créer un conteneur
  - POST `http://localhost/containers`
- Afficher un conteneur
  - GET `http://localhost/containers/3/json`
- Mettre à jour un conteneur
  - PUT `http://localhost/containers/3`
- Supprimer un conteneur
  - DELETE `http://localhost/containers/3`



# L'API

Docker API  
**Docker API - TP**



# DOCKER API - TP

Docker API  
Docker API - TP

## Activer l'API

- Suivez la documentation pour exposer l'API Docker
  - (<https://docs.docker.com/engine/install/linux-postinstall/#configure-where-the-docker-daemon-listens-for-connections>)
- Attention, sous Windows et MacOS la manipulation est différente !



# DOCKER API - TP

Docker API  
Docker API - TP

## Manipuler l'API

- Trouvez quelle version d'API votre daemon Docker utilise et trouvez la doc correspondante sur le site de Docker. Elle vous servira de référence.
- Listez les images présentes localement sur votre poste en interrogeant l'API
- Pulez l'image **ubuntu:latest** en interrogeant l'API
- A l'aide du fichier `create_container_payload_simple.json` fourni par le formateur, créez le conteneur `mysql-test`
- Vérifiez qu'il a bien été créé et démarrez-le via l'API
- Envoyez un `SIGHUP` à votre conteneur puis un `SIGKILL` (encore et toujours via l'API)
- (Bonus 1) En réalité, l'API était déjà exposée sur la machine, par quel mécanisme ?
- (Bonus 2) Lister les images via ce mécanisme
- (Bonus 3) Utiliser `jq` pour lister uniquement le nom des conteneurs via l'API



# L'API

Dans ce chapitre nous avons :

- Découvert l'API de Docker.
- Manipulé l'API de Docker.



# CHAPITRE 9

# **PORTAINER**



# PORTAINER

- Docker Portainer - TP nous fait découvrir Portainer.



# PORTAINER

Docker Portainer - TP



# DOCKER PORTAINER - TP

Docker Portainer - TP

## Installation de Portainer

- La documentation se trouve ici :  
<https://docs.portainer.io/start/install/server/docker/linux>
- Créez un volume nommé portainer\_data
- Lancez Portainer à l'aide de la commande suivante :

```
$ docker run -d -p 8000:8000 -p 9443:9443 --name portainer \
--restart=always \
-v /var/run/docker.sock:/var/run/docker.sock \
-v portainer_data:/data \
portainer/portainer-ce:latest
```



# DOCKER PORTAINER - TP

Docker Portainer - TP

## Lancement de l'agent

- Expliquez la commande
- Listez les conteneurs, que constatez-vous ?
- On ajoute l'agent portainer qui permet entre autre de lister le contenu de volumes docker :

```
$ docker run -d -p 9001:9001 --name portainer_agent --  
restart=always -v /var/run/docker.sock:/var/run/docker.sock -v  
/var/lib/docker/volumes:/var/lib/docker/volumes  
portainer/agent:latest
```



# DOCKER PORTAINER - TP

Docker Portainer - TP

## Résultat

- Ouvrez un navigateur localement et rendez-vous sur <https://localhost:9443>.
- Configurez l'utilisateur administrateur
  - Le nom de ce user est admin par défaut, vous pouvez le changer selon votre convenance et saisir le mot de passe qui lui sera associé.
- Parcourez les menus et essayez de comprendre tout ce qu'il est possible de faire avec cet outil



# PORTAINER

Dans ce chapitre nous avons :

- Découvert Portainer.



# CHAPITRE 10

# **REGISTRY**



# REGISTRY

- **Docker Registry** nous présente la Registry Docker.
- **Docker Registry - TP** nous fait manipuler la Registry.



# REGISTRY

Docker Registry  
Docker Registry - TP



# DOCKER REGISTRY

Docker Registry  
Docker Registry - TP

## Registre

- Exécutez votre propre registre pour stocker et distribuer des images au lieu d'utiliser Docker Hub
- Plusieurs options
  - Exécuter le registre en utilisant le conteneur Registry
  - Des services concurrent (quay.io par exemple)
- Deux versions
  - Registry v1.0 jusqu'à Docker 1.5
  - Registry v2.0 depuis Docker 1.6

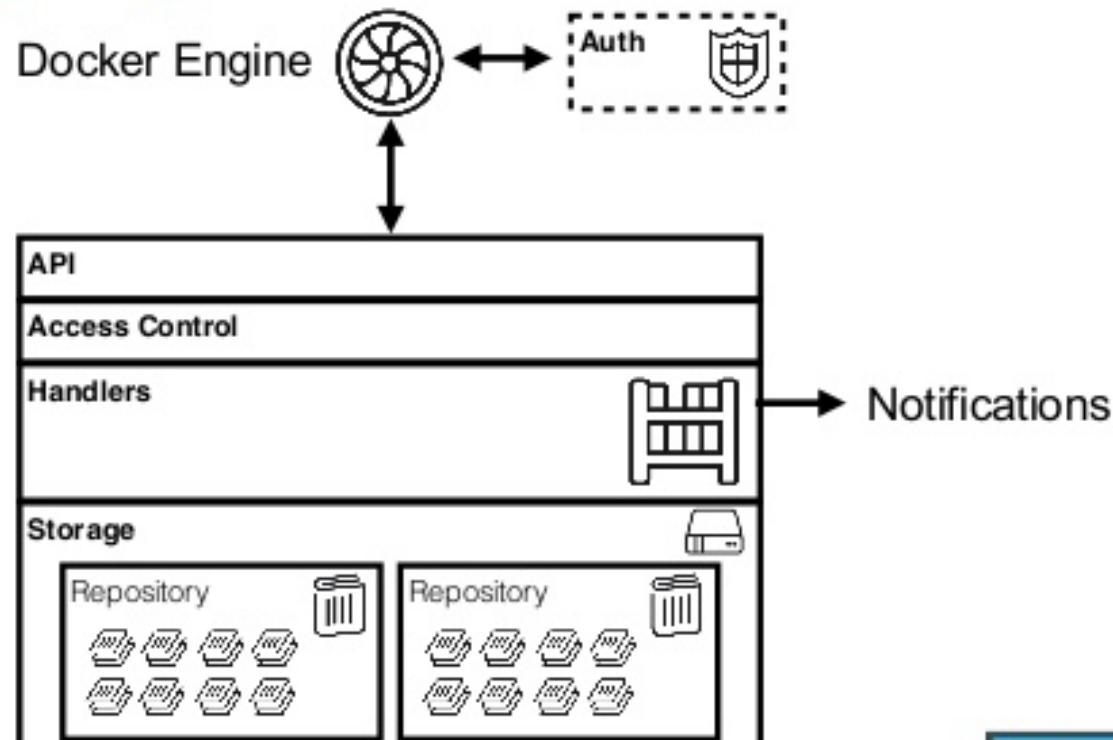


# DOCKER REGISTRY

Docker Registry  
Docker Registry - TP

## Architecture

### Docker Registry 2.0: Architecture





# DOCKER REGISTRY

## Stockage configurable

- Stockage configurable
  - Local disk
  - Amazon S3
  - Microsoft Azure
- Webhooks pour lancer une construction ou envoyer des notifications à certaines personnes lorsque des images ont été poussées
- Accès sécurisé aux images via TLS

Docker Registry  
Docker Registry - TP



# DOCKER REGISTRY

## Public ou privé

- Vous pouvez choisir d'exécuter un registre qui est accessible au public
- Vous pouvez le mettre derrière le pare-feu et le rendre disponible uniquement pour le personnel interne de l'entreprise

Docker Registry  
Docker Registry - TP



# DOCKER REGISTRY

## Configuration d'un registre

- Deux méthodes
  - Utilisez l'image officielle du registre sur le hub docker
  - Téléchargez la source de distribution et créez votre propre image de registre personnalisée
- L'image officielle contient une version pré-configurée du registre v2.0
- L'image officielle est destinée à des fins d'évaluation car sa configuration par défaut ne convient pas à l'utilisation de la production
  - Pas de TLS

```
docker run -d -p 5000:5000 registry:2.0
```



# DOCKER REGISTRY

Docker Registry  
Docker Registry - TP

## Push

- Il faut d'abord créer un tag de l'image avec l'hôte IP ou le domaine du serveur de registre, puis exécuter la commande docker push

```
[root@docker ~]# docker tag 09ea64205e55 myserver.com:5000/masociete/monimage:1.0
```

```
[root@docker ~]# docker push myserver.com:5000/masociete/monimage:1.0
The push refers to a repository [myserver.com:5000/masociete/monimage]
```



# DOCKER REGISTRY

Docker Registry  
Docker Registry - TP

## Lister les tags dans le registre

- Pour lister les tags d'une image, il faut faire une requête du type :  
`<registry host>:<port>/v2/<repo name>/tags/list`

```
[root@docker ~]# curl http://mycompany.com:5000/v2/masociete/monimage/tags/list
{"name": "masociete/monimage", "tags": ["1.0"]}
```



# DOCKER REGISTRY

Docker Registry  
Docker Registry - TP

## Pull

- Pour extraire une image d'un serveur de registre, il faut
  - L' URL du serveur et le port
  - Image repository
  - Image tag



# DOCKER REGISTRY

## Registre non sécurisé

- Par défaut, le démon Docker suppose que tous les registres sont sécurisés et bloque la communication avec des registres non sécurisés
  - Impossible de faire un push ou pull
- Pour permettre une communication avec un registre non sécurisé, le Daemon doit être démarré avec l'option `--insecure-registry` et chaque registre doit être ajouté

```
--insecure-registry myregistry:5000
```



# REGISTRY

Docker Registry  
**Docker Registry - TP**



# DOCKER REGISTRY - TP

Docker Registry  
Docker Registry - TP

## Installation du registre à l'aide de Portainer

- La solution se trouve dans App Templates
  - On peut en installer une sur notre poste de travail simplement via Portainer en allant dans App Templates → Registry
- Il y a maintenant une nouvelle registry disponible dans l'interface, vous avez accès à de nouvelles informations
  - Il faut regarder sur quel port la registry écoute, et c'est ce port que nous allons utiliser dans la suite de ce TP.



# DOCKER REGISTRY - TP

Docker Registry  
Docker Registry - TP

## Manipuler la registry locale

- Pushez dans votre registry locale l'image **ubuntu:latest** (et toutes celles que vous voulez)
- Regardez les logs de la registry pour voir ce qu'il s'est passé (via docker logs ou via Portainer)
- Supprimez l'image **ubuntu:latest** via docker, vérifiez qu'elle n'est bien plus là puis re pullez là depuis votre registry locale



# DOCKER REGISTRY - TP

## L'API de la registry

- La documentation est ici : <https://docs.docker.com/registry/spec/api/>
- Listez le catalogue de repositories de la registry
- Listez les tags d'un repository
- Supprimez une image stockée dans la registry (cette étape pourrait être plus longue que prévue)
  - Indice 1 : Lire la doc ;) "For deletes..."
  - Indice 2 : Trouver dans la doc ce qu'est un digest et comment le récupérer !

Docker Registry  
Docker Registry - TP



# DOCKER REGISTRY - TP

Docker Registry  
Docker Registry - TP

## Exposer sa registry

- Utilisez l'IP de votre machine à la place de localhost pour pousser des images dans la registry locale : que constatez-vous ?  
Ça ne fonctionne pas ! Par défaut docker ne peut pas pousser d'image dans une registry dont l'adresse n'est pas en https (et le certificat valide, etc.)
- Une fois le problème identifié, trouvez dans la documentation comment l'éliminer



# REGISTRY

Dans ce chapitre nous avons :

- Découvert le concept de Registry.
- Utilisé la Registry.



# CHAPITRE 11

# **SWARM**



# SWARM

- **Docker Swarm** nous fait découvrir l'orchestrateur Swarm.
- **Docker Swarm - TP Vagrant** nous permet d'installer un environnement pour tester Swarm.
- **Docker Swarm - TP** nous permet de découvrir les concepts de base de l'orchestration.



# SWARM

**Docker Swarm**

Docker Swarm - TP Vagrant

Docker Swarm - TP



# DOCKER SWARM

## Docker Swarm

Docker Swarm - TP Vagrant  
Docker Swarm - TP

## Swarm

- Docker Swarm est un cluster de hôtes Docker sur lequel on déploie des conteneurs
- Permet de distribuer les charges de travail des conteneurs sur plusieurs machines s'exécutant dans un cluster
- Permet d'assurer une redondance
- L'API de Docker inclut des commandes pour gérer les noeuds du cluster (ajout, suppression des noeuds), et déployer et organiser des services à travers le cluster



# DOCKER SWARM

Docker Swarm

Docker Swarm - TP Vagrant

Docker Swarm - TP

## Modèle

- Pour déployer votre application dans un cluster, vous soumettez un service à un nœud **Manager**. Le nœud **Manager** distribue les unités de travail appelées tasks aux nœuds **Worker**
- Les nœuds de gestion (Manager) effectuent également l'orchestration et le maintient en état souhaité du cluster. Les nœuds Manager élisent un seul Maître pour mener des tâches d'orchestration
- Les nœuds **Worker** reçoivent et exécutent les tâches envoyées depuis les nœuds **Manager**.



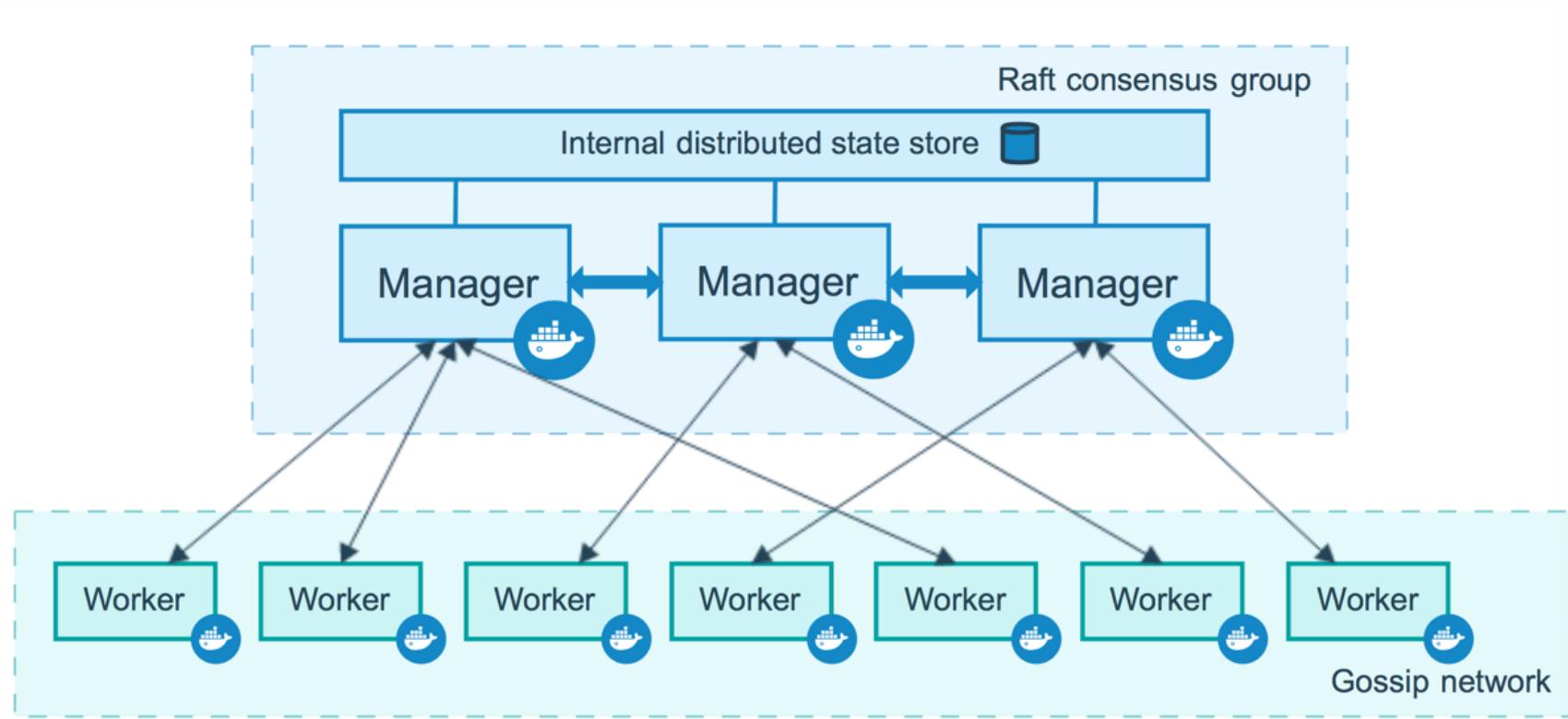
# DOCKER SWARM

Docker Swarm

Docker Swarm - TP Vagrant

Docker Swarm - TP

## Architecture



Thomas Saquet - Docker - Swarm



# DOCKER SWARM

## Agent

- Un agent s'exécute sur chaque nœud Worker et rend-compte des tâches qui lui sont affectées.  
=> Le nœud Worker notifie au nœud Manager l'état actuel de ses tâches afin que le Manager puisse maintenir l'état désiré de chaque Worker.

Docker Swarm

Docker Swarm - TP Vagrant

Docker Swarm - TP



# DOCKER SWARM

## Service

- Un service est la définition des tâches à exécuter sur le nœud Manager ou les nœuds Worker. C'est la structure centrale du système Swarm.
- Lorsque vous créez un service, vous spécifiez l'image du conteneur à utiliser et les commandes à exécuter à l'intérieur des conteneurs en cours d'exécution.

Docker Swarm

Docker Swarm - TP Vagrant

Docker Swarm - TP



# DOCKER SWARM

## Modèle

- Dans le modèle de services Replicas, le Manager distribue un nombre de copie spécifié de tâches aux Workers
- Dans le modèle de services Global, Swarm exécute une tâche pour le service sur chaque nœud disponible dans le cluster.

Docker Swarm

Docker Swarm - TP Vagrant

Docker Swarm - TP



# DOCKER SWARM

Docker Swarm

Docker Swarm - TP Vagrant

Docker Swarm - TP

## Modèle

- Une tâche comporte un conteneur Docker et les commandes à exécuter à l'intérieur du conteneur. C'est l'unité de planification atomique de Swarm.
- Les nœuds Manager attribuent des tâches aux nœuds du travail en fonction du nombre de répliques définis dans l'option scale du service.
- Une fois qu'une tâche est attribuée à un nœud, elle ne peut pas se déplacer vers un autre nœud. Il ne peut fonctionner que sur le nœud assigné ou échouer.



# DOCKER SWARM

## Docker Swarm

Docker Swarm - TP Vagrant  
Docker Swarm - TP

## Création du cluster Swarm

- La commande suivante permet de créer un nouveau cluster swarm :

```
docker swarm init --advertise-addr <MANAGER-IP>
```

```
docker@node1:~$ docker swarm init --advertise-addr 192.168.99.100
```

```
Swarm initialized: current node (f9js2mda7uf70bqydetmcgo6p) is now a manager.
```

```
To add a worker to this swarm, run the following command:
```

```
docker swarm join --token SWMTKN-1-3716t7lma6yeexj1mirjhnma2yh5ydrft5 192.168.99.100:2377
```

```
To add a manager to this swarm, run 'docker swarm join-token manager' and follow the instructions.
```



# DOCKER SWARM

Docker Swarm

Docker Swarm - TP Vagrant

Docker Swarm - TP

## Etat du cluster Swarm

La commande suivante permet de visualiser l'état du cluster swarm :

```
docker@node1:~$ docker info
Logging Driver: json-file
Cgroup Driver: cgroupfs
Plugins:
Volume: local
Network: bridge host macvlan null overlay
Log: awslogs fluentd gcplogs gelf journald json-file logentries splunk syslog
Swarm: active
NodeID: f9js2mda7uf70bqydetmcgo6p
Is Manager: true
ClusterID: xh1d7tk4ja3qgk5nf5nljvjp1
Managers: 1
Nodes: 1
Orchestration:
Task History Retention Limit: 5
```



# DOCKER SWARM

## Docker Swarm

Docker Swarm - TP Vagrant  
Docker Swarm - TP

## Ajout des workers

- La commande suivante permet d'ajouter un Worker dans le cluster swarm :  
`docker swarm join --token <token> IP-Manager`

```
docker@node2:~$ docker swarm join --token SWMTKN-1-3716t7lm29 192.168.99.100:2377
This node joined a swarm as a worker.
```

```
docker@node1:~$ docker node ls
```

ID	HOSTNAME	STATUS	AVAILABILITY	MANAGER	STATUS
f9js2mda7uf70bqydetmcgo6p	* node1	Ready	Active	Leader	
2bkuf00chye6uy4cebr2728jr	node2	Ready	Active		



# DOCKER SWARM

## Docker Swarm

Docker Swarm - TP Vagrant  
Docker Swarm - TP

## Déployer un service

- La commande suivante permet de déployer un service dans le cluster swarm avec le nombre de répliques désiré :

```
docker service create --replicas <#> Image
```

```
docker@node1:~$ docker service create --replicas 2 --name ping alpine ping 127.0.0.1
loemi7bv1kne7wo2srkf04kiv
```

```
docker@node1:~$ docker service ls
ID          NAME MODE      REPLICAS  IMAGE          PORTS
loemi7bv1kne ping replicated 2/2    alpine:latest
```



# DOCKER SWARM

Docker Swarm  
Docker Swarm - TP Vagrant  
Docker Swarm - TP

## Inspecter un service

- docker service inspect --pretty <service>

```
docker@node1:~$ docker service inspect --pretty ping

ID: loemi7bv1kne7wo2srkf04kiv
Name: ping
Service Mode: Replicated
Replicas: 2
Placement:
UpdateConfig:
Parallelism: 1
On failure: pause
Monitoring Period: 5s
Max failure ratio: 0
```



# DOCKER SWARM

## Lister les services

- docker service ps <service>

```
docker@node1:~$ docker service ps ping
```

ID	NAME	IMAGE	NODE	DESIRED STATE	CURRENT STATE	ERROR PORTS
nd9ntu992wex	ping.1	alpine:latest	node1	Running	Running 10 minutes ago	
acixdloq7zrq	ping.2	alpine:latest	node2	Running	Running 10 minutes ago	

Docker Swarm

Docker Swarm - TP Vagrant

Docker Swarm - TP



# DOCKER SWARM

Docker Swarm

Docker Swarm - TP Vagrant

Docker Swarm - TP

## Modifier le nombre de replicas

- docker service scale service=#

```
docker@node1:~$ docker service scale ping=4
ping scaled to 4
```

```
docker@node1:~$ docker service ps ping
```

ID	NAME	IMAGE	NODE	DESIRED STATE	CURRENT STATE	ERROR PORTS
nd9ntu992wex	ping.1	alpine:latest	node1	Running	Running 10 minutes ago	
acixdloq7zrq	ping.2	alpine:latest	node2	Running	Running 10 minutes ago	
js2ubbkubb8a	ping.3	alpine:latest	node3	Running	Preparing 3 seconds ago	
64gko3invol	ping.4	alpine:latest	node3	Running	Preparing 3 seconds ago	



# DOCKER SWARM

Docker Swarm  
Docker Swarm - TP Vagrant  
Docker Swarm - TP

## Noeud en état "drain"

- docker node update --availability drain <service>

```
docker@node1:~$ docker node update --availability drain node2
Node2
docker@node1:~$ docker service ps ping
ID           NAME      IMAGE      NODE  DESIRED STATE  CURRENT STATE    ERROR PORTS
nd9ntu992wex ping.1  alpine:latest node1 Running   Running  20 minutes ago
jbr3gzlmywgc ping.2  alpine:latest node1 Running   Running  10 seconds ago
acixdloq7zrq \_ ping.2 alpine:latest node2 Shutdown  Shutdown 10 seconds ago
js2ubbkubb8a ping.3  alpine:latest node3 Running   Running  5 minutes ago
64gko3invol ping.4  alpine:latest node3 Running   Running  5 minutes ago
```



# DOCKER SWARM

## Docker Swarm

Docker Swarm - TP Vagrant  
Docker Swarm - TP

## Noeud en état "active"

- docker node update --availability active <node>

```
docker@node1:~$ docker node update --availability active node2
Node2
```

```
docker@node1:~$ docker service ps ping
ID          NAME      IMAGE      NODE  DESIRED STATE  CURRENT STATE    ERROR PORTS
adj283f3acaq ping.1  alpine:latest node1 Running        Running 48 seconds ago
sed6bo01e5o5 \_ping.1 alpine:latest node2 Shutdown       Shutdown 50 seconds ago
rqcjsjz9qaxn ping.2  alpine:latest node1 Running        Running 2 minutes ago
4jt9khyb4exe ping.3  alpine:latest node3 Running        Running about a minute ago
1522wwvt3ol8 ping.4  alpine:latest node3 Running        Running about a minute ago
```



# DOCKER SWARM

## Docker Swarm

Docker Swarm - TP Vagrant  
Docker Swarm - TP

## Noeud en état "pause"

- docker node update --availability pause <node>

```
docker@node1:~$ docker node update --availability pause node2
Node2
```

```
docker@node1:~$ docker node ls
ID                      HOSTNAME  STATUS  AVAILABILITY  MANAGER STATUS ENGINE VERSION
Upa8ru65qmpdu4vpe03qqr0u1 *  node1    Ready   Active        Leader      &nbsp;  18.05.0-ce
oz3yp2dmri87hb8gnvpsqvi0h   node2    Ready   Pause         &nbsp;      18.05.0-ce
t9rhmo12lne238hngrq3j95vj   node3    Ready   Active        &nbsp;      18.05.0-ce
```



# SWARM

Docker Swarm

**Docker Swarm - TP Vagrant**

Docker Swarm - TP



# DOCKER SWARM - TP VAGRANT

Docker Swarm  
Docker Swarm - TP Vagrant  
Docker Swarm - TP

## Déploiement des VMs avec Vagrant + VirtualBox

- Ce « TP » est uniquement là pour permettre de faire un TP sur Swarm ensuite. Il a juste vocation à créer des VMs automatiquement avec Vagrant pour pouvoir administrer plusieurs nœuds avec Swarm.
- Toutes les commandes à taper sont là, il n'y a rien à chercher en plus.
- Vagrant est un logiciel libre et open-source pour la création et la configuration des environnements de développement virtuel. Il peut être considéré comme un wrapper autour de logiciels de virtualisation comme VirtualBox.



# DOCKER SWARM - TP VAGRANT

Docker Swarm  
**Docker Swarm - TP Vagrant**  
Docker Swarm - TP

## Installation de Vagrant

- Installez Vagrant comme indiqué dans la documentation :  
<https://www.vagrantup.com/downloads>
- Ci-dessous les commandes qui correspondent à Debian / Ubuntu, pour une autre distribution utilisez la documentation !

```
$ curl -fsSL https://apt.releases.hashicorp.com/gpg | sudo apt-key add -
$ sudo apt-add-repository "deb [arch=amd64] https://apt.releases.hashicorp.com
$(lsb_release -cs) main"
$ sudo apt-get update && sudo apt-get install vagrant
```



# DOCKER SWARM - TP VAGRANT

Docker Swarm  
**Docker Swarm - TP Vagrant**  
Docker Swarm - TP

## Provisionnement des VMs

- Clonez le repo <https://github.com/tsaquet/vagrant-docker-swarm> et rentrez dedans.
- Lancez la commande vagrant up qui va provisionner 3 VMs Ubuntu Bionic via VirtualBox avec docker d'installé.
- Attendez la fin de l'exécution de Vagrant et vérifiez que les VMs sont bien là :

```
$ vagrant status
Current machine states:
manager running (virtualbox)
worker1 running (virtualbox)
worker2 running (virtualbox)
```



# DOCKER SWARM - TP VAGRANT

Docker Swarm  
Docker Swarm - TP Vagrant  
Docker Swarm - TP

Pour se connecter aux différentes VMs, il suffit de faire

```
vagrant ssh <manager|worker1|worker2>
```



# SWARM

Docker Swarm

Docker Swarm - TP Vagrant

**Docker Swarm - TP**



# DOCKER SWARM - TP

Docker Swarm  
Docker Swarm - TP Vagrant  
**Docker Swarm - TP**

## Initialiser les premiers noeuds

- Entrez dans la VM Manager
- Initialisez le premier nœud en tant que manager
- Initialisez les autres nœuds en tant que workers
- Faites la promotion des nœuds workers en nœuds managers



# DOCKER SWARM - TP

Docker Swarm  
Docker Swarm - TP Vagrant  
**Docker Swarm - TP**

## Création de services

- Créez un service web de type replicated avec 2 replicas, basé sur un service httpd
- Faites évoluer le service web à 4 replicas
- Créez un service debug de type global, basé sur alpine



# DOCKER SWARM - TP

Docker Swarm  
Docker Swarm - TP Vagrant  
**Docker Swarm - TP**

## Service visualizer

- Visualizer est un outil graphique qui permet de visualiser les services sur un cluster SWARM.
- Vous trouverez ce service à l'adresse suivante :  
<https://hub.docker.com/r/dockersamples/visualizer>
- Déployez-le en tant que service appelé viz & consultez-le dans votre navigateur.



# DOCKER SWARM - TP

Docker Swarm  
Docker Swarm - TP Vagrant  
**Docker Swarm - TP**

## Cas de panne

- Pour simuler une panne, arrêtez le manager
- Redémarrez le noeud et regardez ce qu'il se passe



# DOCKER SWARM - TP

Docker Swarm  
Docker Swarm - TP Vagrant  
**Docker Swarm - TP**

## Docker-compose en mode swarm

- Créez une pile compose/swarm pour lancer un service global debug basé sur alpine, et un service replicated avec 4 replicas basé sur httpd
- Lancez la pile & visualisez le résultat



# SWARM

Dans ce chapitre nous avons :

- Découvert Swarm.
- Installé un environnement pour tester Swarm.
- Vu ce qu'était l'orchestration grâce à Docker Swarm.



CHAPITRE 12

# ADMINISTRATION



# ADMINISTRATION

- **Administration de Docker** nous présentera les bases de l'administration de Docker.
- **Sécurité TLS** où l'on parle sécurité.



# ADMINISTRATION

Administration de Docker  
Sécurité TLS



# ADMINISTRATION DE DOCKER

## Configuration du démon

- La configuration du démarrage et arrêt du démon Docker dépend d'un certain nombre de facteurs
  - Le démon s'exécute comme un service ?
  - Quelle distribution Linux
    - service
    - systemctl
- Exécution en mode interactif en arrière plan

Administration de Docker  
Sécurité TLS



# ADMINISTRATION DE DOCKER

## Exécution en mode service

- Ubuntu et Debian
  - Utiliser la commande service
    - sudo service docker stop
    - sudo service docker start
    - sudo service docker restart

Administration de Docker  
Sécurité TLS



# ADMINISTRATION DE DOCKER

## Exécution en mode service

- RHEL & CentOS
  - Utiliser la commande systemctl
    - systemctl start docker
    - systemctl stop docker
    - systemctl restart docker
  - Dans les nouvelles version la commande service est une redirection vers la commande systemctl

Administration de Docker  
Sécurité TLS



# ADMINISTRATION DE DOCKER

## Exécution interactive

- S'il ne fonctionne pas en tant que service, lancez le démon Docker
  - sudo docker daemon & (avant la 1.12)
  - sudo dockerd & (depuis la 1.12)
- S'il ne fonctionne pas en tant que service, envoyez le signal SIGTERM pour arrêter le démon
  - sudo kill \$(pidof docker)

Administration de Docker  
Sécurité TLS



# ADMINISTRATION DE DOCKER

## Fichiers de configuration

- Ubuntu et Debian
  - /etc/default/docker
  - La variable DOCKER\_OPTS est utilisée pour ajouter des options au démon s'il est lancé comme un service
- Il faut relancer le service pour prendre en compte les modifications
  - sudo service docker restart

Administration de Docker  
Sécurité TLS



# ADMINISTRATION DE DOCKER

## Fichiers de configuration

- RHEL et Centos

- Le mécanisme systemd est utilisé pour exécuter le démon
- Les options de lancement sont dans le fichier docker.service
- Le fichier docker.service se trouve dans /usr/lib/systemd/system ou /etc/systemd/service
- Exécuter la commande : find / -name docker.service

```
[root@docker ~]# find / -name docker.service
/etc/systemd/system/multi-user.target.wants/docker.service
/usr/lib/systemd/system/docker.service
```



# ADMINISTRATION DE DOCKER

## Fichiers de configuration

Administration de Docker  
Sécurité TLS

```
[root@localhost ~]# cat /usr/lib/systemd/system/docker.service
[...]
[Service]
Type=notify
# the default is not to use systemd for cgroups because the delegate issues still
# exists and systemd currently does not support the cgroup feature set required
# for containers run by docker
ExecStart=/usr/bin/dockerd -H fd:// --containerd=/run/containerd/containerd.sock
ExecReload=/bin/kill -s HUP $MAINPID
TimeoutSec=0
RestartSec=2
Restart=always
```



# ADMINISTRATION DE DOCKER

## Les options

- Lancer le démon en mode réseau.
  - Le démon écoute sur une socket TCP
- Activer le mode debug
  - Valider le niveau de journalisation
- Spécifier un serveur DNS
- Ajouter des registres non sécurisés
- Valider la sécurité du démon avec TLS

Administration de Docker  
Sécurité TLS



# ADMINISTRATION DE DOCKER

## Journalisation

- Démarrez le démon avec l'option `--log-level` et spécifiez le niveau de journalisation
  - Debug
  - Info
  - Warn
  - Error
  - Fatal

Administration de Docker  
Sécurité TLS



# ADMINISTRATION DE DOCKER

## Journalisation (RHEL & Centos)

- Sur les systèmes basés sur systemd la journalisation est gérée par le démon journald
- La commande journalctl permet de visualiser le log
- Utiliser l'option -u pour filtrer par service journalctl -u docker.service

```
[root@docker ~]# journalctl -u docker.service
-- Logs begin at Tue 2022-11-16 15:50:22 EST, end at Tue 2022-11-16 16:02:13 EST. --
nov. 16 16:02:05 localhost.localdomain systemd[1]: Starting Docker Application Container Engine...
nov. 16 16:02:06 localhost.localdomain dockerd[3269]: time="2022-11-16T16:02:06.427139078-05:00" level=info msg="Starting up"
nov. 16 16:02:06 localhost.localdomain dockerd[3269]: time="2022-11-16T16:02:06.429055055-05:00" level=info msg="parsed scheme: \"unix\"" module=grpc
nov. 16 16:02:06 localhost.localdomain dockerd[3269]: time="2022-11-16T16:02:06.429072918-05:00" level=info msg="scheme \"unix\" not registered, fallback to default scheme" module=grpc
nov. 16 16:02:06 localhost.localdomain dockerd[3269]: time="2022-11-16T16:02:06.429123794-05:00" level=info msg="ccResolverWrapper: sending update to cc: {[{unix:///run/containerd/containerd.sock <nil>} 0 <nil>]}
nov. 16 16:02:06 localhost.localdomain dockerd[3269]: time="2022-11-16T16:02:06.429143759-05:00" level=info msg="ClientConn switching balancer to \"pick_first\"" module=grpc
nov. 16 16:02:06 localhost.localdomain dockerd[3269]: time="2022-11-16T16:02:06.432558588-05:00" level=info msg="parsed scheme: \"unix\"" module=grpc
nov. 16 16:02:06 localhost.localdomain dockerd[3269]: time="2022-11-16T16:02:06.432572225-05:00" level=info msg="scheme \"unix\" not registered, fallback to default scheme" module=grpc
nov. 16 16:02:06 localhost.localdomain dockerd[3269]: time="2022-11-16T16:02:06.432582957-05:00" level=info msg="ccResolverWrapper: sending update to cc: {[{unix:///run/containerd/containerd.sock <nil>} 0 <nil>]}
nov. 16 16:02:06 localhost.localdomain dockerd[3269]: time="2022-11-16T16:02:06.432589294-05:00" level=info msg="ClientConn switching balancer to \"pick_first\"" module=grpc
nov. 16 16:02:06 localhost.localdomain dockerd[3269]: time="2022-11-16T16:02:06.452689750-05:00" level=info msg="[graphdriver] using prior storage driver: overlay2"
nov. 16 16:02:06 localhost.localdomain dockerd[3269]: time="2022-11-16T16:02:06.591253854-05:00" level=warning msg="Your kernel does not support cgroup blkio weight"
nov. 16 16:02:06 localhost.localdomain dockerd[3269]: time="2022-11-16T16:02:06.591274584-05:00" level=warning msg="Your kernel does not support cgroup blkio weight_device"
nov. 16 16:02:06 localhost.localdomain dockerd[3269]: time="2022-11-16T16:02:06.591395630-05:00" level=info msg="Loading containers: start."
```

Administration de Docker  
Sécurité TLS



# ADMINISTRATION DE DOCKER

## Démon Docker en réseau

- Par défaut, le client Docker et le démon sont sur le même hôte
- Pour connecter le client à un démon Docker exécuté sur un hôte différent, il faut :
  - Tout d'abord, le démon docker doit écouter sur une socket réseau (TCP)
  - Pour des raisons de sécurité, le démon peut utiliser une socket chiffrée. Il faudra configurer TLS
  - Modifier le client pour se connecter au démon distant

Administration de Docker  
Sécurité TLS



# ADMINISTRATION DE DOCKER

Administration de Docker  
Sécurité TLS

## Erreur de connexion

- Le type message d'erreur ci-dessous signifie généralement
  - Le démon Docker ne fonctionne pas
  - Problème de permission de connexion au démon docker
  - Votre client Docker essaie de se connecter au démon en utilisant la socket, mais le démon ne l'écoute pas
  - Vous n'utilisez pas TLS pour vous connecter au démon

```
[root@docker ~]# docker info
Cannot connect to the Docker daemon at unix:///var/run/docker.sock. Is the docker daemon running?
```



# ADMINISTRATION DE DOCKER

Administration de Docker  
Sécurité TLS

## Ecoute sur une socket TCP

- On peut configurer le démon Docker pour écouter sur une socket TCP, en utilisant l'option `--host` ou `-H` et spécifions l'adresse TCP et le port
- Pour l'adresse, vous pouvez spécifier une adresse IP pour écouter ou spécifier `0.0.0.0` pour écouter toutes les adresses réseau
- Le port `2375` pour une communication non chiffrée
- Le port `2376` pour une communication chiffrée



# ADMINISTRATION DE DOCKER

## Ecoute sur la socket TCP

- La socket TCP écoute toutes les adresses réseau

```
dockerd -H tcp://0.0.0.0:2375
```

- La socket TCP écoute une adresse spécifique

```
dockerd -H tcp://10.2.1.1:2375
```

Administration de Docker  
Sécurité TLS



# ADMINISTRATION DE DOCKER

## Connexion du client

- Par défaut, le client Docker suppose que le démon écoute sur une socket Unix
- Il faut configurer le client pour se connecter à un démon docker distant
  - Utiliser l'option -H de la commande docker
  - Configurer la variable d'environnement DOCKER\_HOST

Administration de Docker  
Sécurité TLS



# ADMINISTRATION DE DOCKER

## Ecoute sur plusieurs sockets

- Le démon Docker peut écouter à la fois une socket Unix et une socket TCP
- On peut utiliser l'option -H plusieurs fois
  - Pour la socket unix
    - unix:///var/run/docker.sock
  - Pour la socket réseau
    - tcp://0.0.0.0:2376

Administration de Docker  
Sécurité TLS



# ADMINISTRATION

Administration de Docker  
**Sécurité TLS**



# SÉCURITÉ TLS

Administration de Docker  
Sécurité TLS

## Conteneurs et sécurité

- Docker permet de rendre les applications plus sûres car il fournit un ensemble réduit de privilèges
- Les espaces de noms (namespaces) fournissent une vue isolée du système.  
Chaque conteneur a son propre :
  - IPC (inter-process communication), stack TCP/IP, file system / etc...
- Les processus s'exécutant dans un conteneur ne peuvent pas voir les processus d'un autre conteneur
- Les groupes de contrôle (Cgroups) isolent l'utilisation des ressources par conteneur
- S'assure qu'un conteneur compromis ne fera pas tomber l'hôte entier en épuisant les ressources



# SÉCURITÉ TLS

## Attention

- Le démon Docker doit fonctionner en tant que root
- Assurez-vous que seuls les utilisateurs de confiance peuvent contrôler le démon Docker
- Qui fait partie du groupe de docker ?
- Si vous liez le démon à une socket TCP, sécurisez-le avec TLS

Administration de Docker  
Sécurité TLS



# SÉCURITÉ TLS

Administration de Docker  
Sécurité TLS

## TLS Transport Layer Security

- Evolution de SSL
- Utilise la cryptographie à clé publique pour chiffrer les connexions
- Les clés sont signées avec des certificats gérés par un tiers de confiance.
- Ces certificats attestent l'identité du serveur
- Chaque transaction est donc chiffrée et authentifiée



# SÉCURITÉ TLS

## TLS pour Docker

- Docker fournit des mécanismes pour authentifier à la fois le serveur et le client
- Fournit une authentication forte, une autorisation et un chiffrement pour toute connexion API sur le réseau
- Les clés client peuvent être distribuées aux clients autorisés
- Pré-requis
  - OpenSSL installé
  - Un dossier pour stocker les clés protégé en mode 700

Administration de Docker  
Sécurité TLS



# SÉCURITÉ TLS

Administration de Docker  
Sécurité TLS

## Processus de configuration de TLS

- Créer l'autorité de certification (CA)
  - Besoin d'une clé privée et d'un certificat CA
- Configurer la clé privée du serveur
- Créer une demande de signature de certificat (CSR) pour le serveur
- Signer la clé du serveur avec le CSR par rapport à notre CA
- Créer une clé privée client et CSR
- Signer la clé du client avec le CSR par rapport à notre autorité de certification
- Exécuter le démon Docker avec TLS activé et spécifiez l'emplacement de la clé privée de l'autorité de certification, du certificat de serveur et de la clé du serveur
  - Et le configurer pour écouter sur TCP
- Pointer le client Docker vers l'adresse TCP du démon et spécifiez l'emplacement du certificat client et la clé en tant que clé privée de l'autorité de certification



# ADMINISTRATION

Dans ce chapitre nous avons :

- Les bases de l'administration de Docker.
- Abordé les notions de sécurité TLS.



# FIN

Merci pour votre attention !