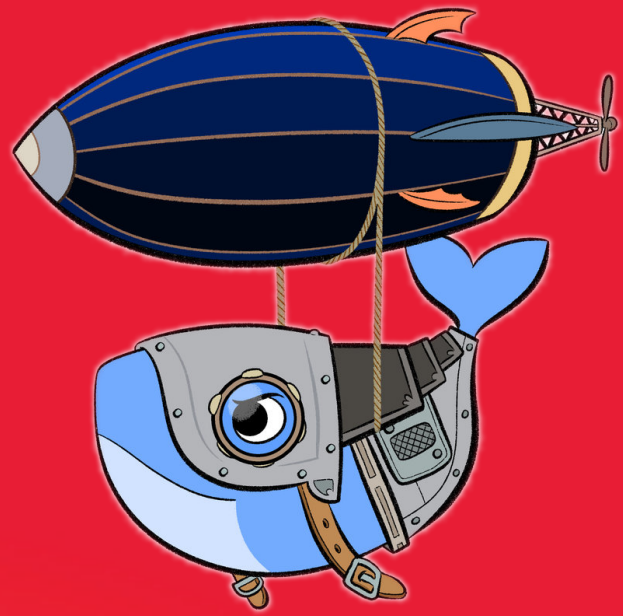
Expérimentations Spark sur cluster privé : Retour sur 3 ans d'utilisation de Swarm

Thomas Calmant INRIA Grenoble Rhône-Alpes SED/Tyrex 14 Mars 2019

1 Contexte

(en 2016)



Contexte

- Projet Datalyse
  - Pipeline de traitement de données de sources hétérogènes
  - Basé sur Cloudera (Hadoop, HDFS, ...), Silk, ...
  - Exécution de code Java et de scripts bash

- Projet SparqlGX
  - Base de données sémantiques (RDF) stockée sur HDFS
  - Requêtage SPARQL optimisé et distribué par Spark
  - Code écrit en OCaml et Scala

Infrastructure initiale

- 2 machines physiques
  - 1 grappe de 6 VMs
  - 1 grappe de 12 VMs

Infrastructure initiale

- 2 machines physiques
  - 1 grappe de 6 VMs
  - 1 grappe de 12 VMs

- Inconvénients visibles
  - Lenteur des E/S
  - Consommation inutile de ressources (OS, services, ...)
  - Contraintes de versions d'OS et de bibliothèques dans les VMs Cloudera

Infrastructure initiale

- 2 machines physiques
  - 1 grappe de 6 VMs
  - 1 grappe de 12 VMs

- Inconvénients visibles
  - Lenteur des E/S
  - Consommation inutile de ressources (OS, services, ...)
  - Contraintes de versions d'OS et de bibliothèques dans les VMs Cloudera

Réception de 3 nouveaux serveurs à la fin du projet Datalyse

Comment intégrer ces 3 machines ?

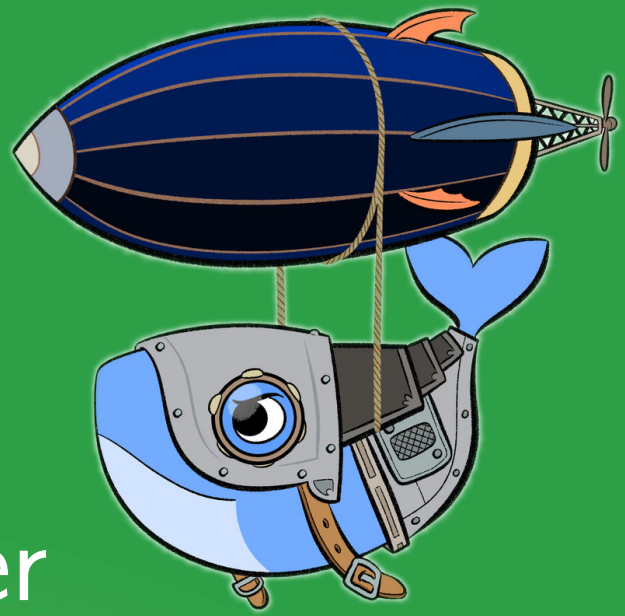
- X Faire 3 nouvelles grappes (aucun intérêt)
- X Héberger 1 grappe sur plusieurs serveurs
  (ne résout pas les inconvénients)
- X Utilisation directe des hôtes
  (ouvre les portes du Tartare)
- V Tester une infrastructure à conteneur en parallèle de l'infrastructure actuelle

Comment intégrer ces 3 machines ?

- *X* Faire 3 nouvelles grappes (aucun intérêt)
- *X* Héberger 1 grappe sur plusieurs serveurs
  (ne résout pas les inconvénients)
- *X* Utilisation directe des hôtes
  (ouvre les portes du Tartare)
- *V* Tester une infrastructure à conteneur en parallèle de l'infrastructure actuelle

- Solution retenue :
  1. Choisir entre Docker et rkt (seules options *viables* en 2016)
  2. Tester les conteneurs sur 1 machine
  3. Tester les conteneurs sur 2 machines
  4. Déployer sur le cluster entier, remis à zéro

2
Mise en place de Docker



Adoption de Docker par étapes

Étape 1 : Essais sur un seul serveur

1. Réutilisation d'images publiques
2. Création d'images spécifiques
   2.1 Images "tout-en-un"
   2.2 1 image par service
3. Liaison et isolation des conteneurs

Adoption de Docker par étapes

Étape 1 : Essais sur un seul serveur

1. Réutilisation d'images publiques
2. Création d'images spécifiques
   - 2.1 Images "tout-en-un"
   - 2.2 1 image par service
3. Liaison et isolation des conteneurs
   - 3.1 À la main

```
docker run --link mysql:db …
docker run -p 8080:80 …
```

Adoption de Docker par étapes

Étape 1 : Essais sur un seul serveur

1. Réutilisation d'images publiques
2. Création d'images spécifiques
   - 2.1 Images "tout-en-un"
   - 2.2 1 image par service
3. Liaison et isolation des conteneurs
   - 3.1 À la main
   - 3.2 Avec Docker Compose

```
vim docker-compose.yml
docker-compose up -d
```

Adoption de Docker par étapes

Étape 2 : Essais entre deux serveurs

1. À la main : ligne de commande et ouverture de ports

```
docker run -p 7077:7077 …
```

Adoption de Docker par étapes

Étape 2 : Essais entre deux serveurs

1. À la main : ligne de commande et ouverture de ports
2. Avec Docker Compose et ouverture de ports

```
docker-compose up -d -f cluster_compose.yml
```

Adoption de Docker par étapes

Étape 2 : Essais entre deux serveurs

1. À la main : ligne de commande et ouverture de ports
2. Avec Docker Compose et ouverture de ports
3. Avec Docker Swarm
   3.1 En ligne de commande (services, utilisées en 2016 et 2017)

```
docker service create --name spark_master …
```

Adoption de Docker par étapes

Étape 2 : Essais entre deux serveurs

1. À la main : ligne de commande et ouverture de ports
2. Avec Docker Compose et ouverture de ports
3. Avec Docker Swarm
   - 3.1 En ligne de commande (services, utilisées en 2016 et 2017)
   - 3.2 Avec les *stacks* (compositions, utilisées à partir de 2017)

```
docker stack deploy -c cluster_stack.yml cluster
```

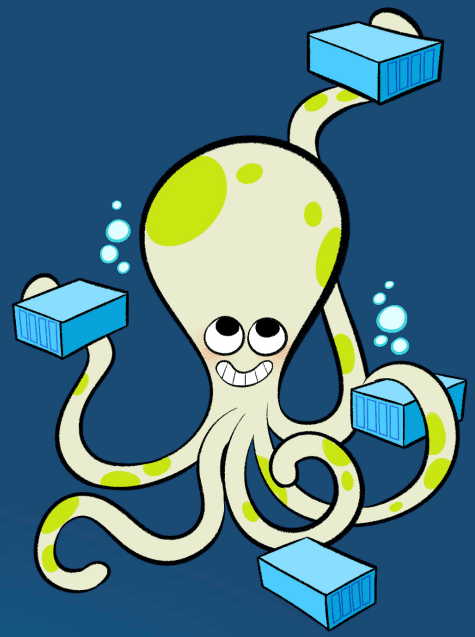
Adoption finale de Docker

1. Remise à zéro des serveurs :
   - Passage à Ubuntu 16.04 LTS
   - Mise en place des mise à jour automatiques
   - Installation de Docker (et rien d'autre)

2. Mise en place du Docker Swarm :
   - Définition du Swarm Manager (machine avec le plus de RAM)
   - Ajout des autres machines comme Workers

3. Déploiement des services
   - Initialement, par un jeu de scripts
   - Aujourd'hui, par des *stacks*

3
Infrastructure actuelle
(depuis 2018)



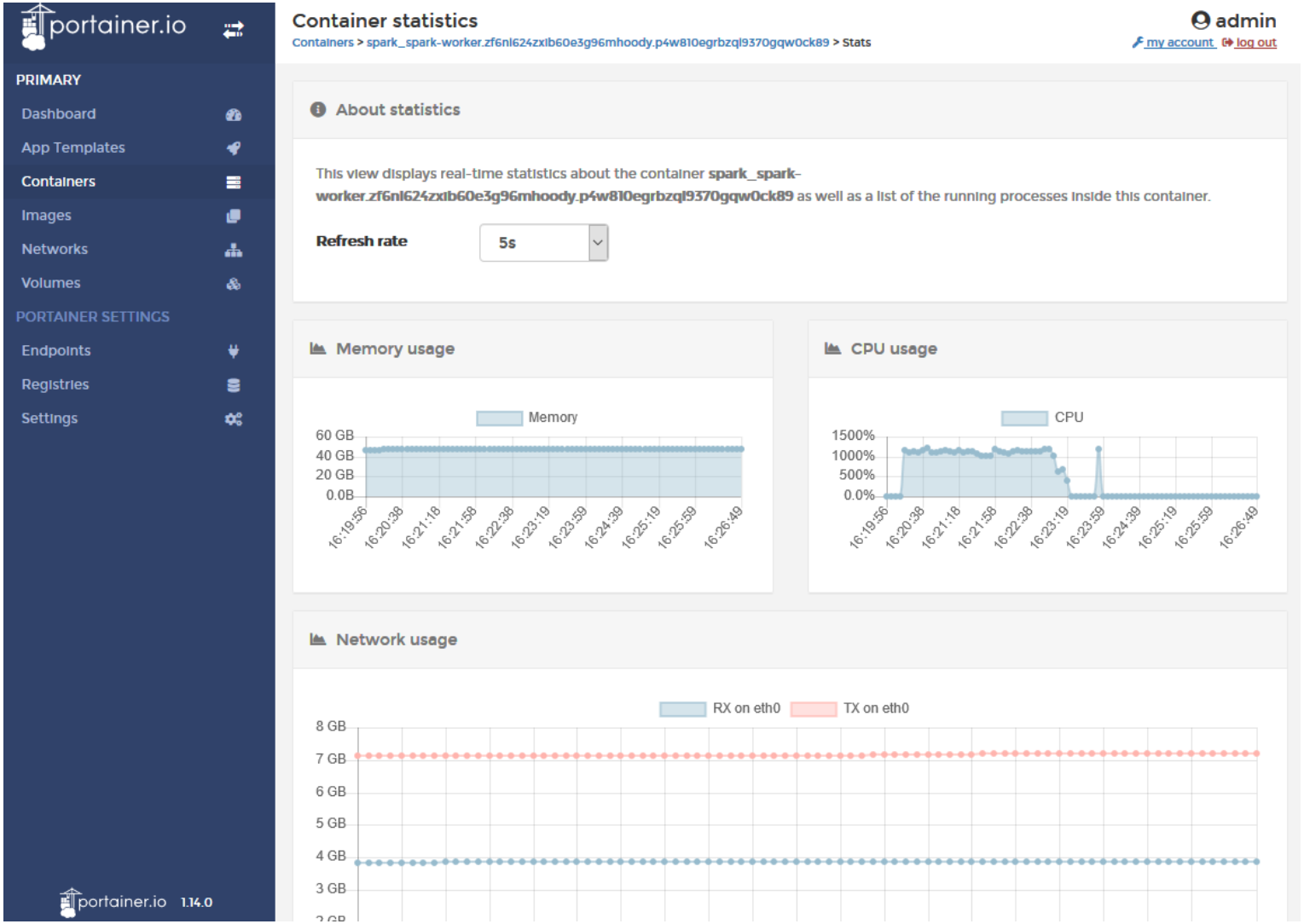
État du cluster

- 6 serveurs de calculs, dont :
  - 2 Swarm Managers
  - 1 serveur avec GPU
  - 1 serveur de données partagées par NFS
  - 1 serveur contenant réplication des données (non partagée)
- Configuration commune des *stacks*
  - Volumes NFS (avec le plugin NetShare)
  - Réseau inter-machine *overlay* (virtuel, chiffré, géré par Swarm)
- 3 *stacks* définissent l'état standard du cluster
  - d'autres *stacks* sont ajoutées selon les besoins

Stack 1 : Portainer

- Portainer : interface web de gestion de Docker
  - Surveillance / accès simplifié aux conteneurs
  - Accès facile aux statistiques de consommation ressources
- 1 instance de Portainer par hôte physique
- Accès par un port TCP ouvert sur chaque hôte
- Accès protégé par mot de passe donné en paramètre (encodé)

Stack 1 : Portainer



Stack 1 : Portainer

```
version: '3.2'
services:
  portainer:
    image: portainer/portainer
    command: [--admin-password, "...", -H,
    ↪  "unix:///var/run/docker.sock"]
    volumes:
      - /var/run/docker.sock:/var/run/docker.sock
    ports:
      - target: 9000
        published: 9000
        protocol: tcp
        mode: host
    deploy:
      mode: global
```

Stack 2 : Registre d'images Docker

- Stockage des images Docker personnalisées
- Accès par le VLAN d'équipe et le cluster *via* le frontal HTTPS

Service	Placement	Volumes
nginx	Fixé	Fichiers de configuration
registry	Fixé	Dossier local de stockage

Stack 2 : Registre d'images Docker

```
version: '3.2'
services:
  nginx:
    image: nginx
    ports:
      - target: 443
        published: 443
        protocol: tcp
        mode: host
    volumes:
      - ./nginx:/etc/nginx/conf.d
    deploy:
      placement:
        constraints:
          - node.hostname == serv6
```

```
registry:
    image: registry
    environment:
      REGISTRY_STORAGE_FILESYSTEM_ROOTDIRECTORY:
      ↪  /data
    volumes:
      - ./data:/var/lib/registry
      - ./conf.yml:/etc/docker/registry/config.yml
    deploy:
      placement:
        constraints:
          - node.hostname == serv6
```

Stack 3 : Spark

- Principale *stack* d'expérimentation
- Utilisation du plugin Netshare pour les volumes NFS
  - Volume commun (données et notebooks d'équipe)
  - Volume "stagiaires"
- Accès réseau :
  - Accès externe direct aux notebooks
  - Accès *via* un proxy à Spark et HDFS

Stack 3 : Spark

Service	Placement	Volumes
Spark Master	Instance unique	NFS commun NFS stagiaires
Spark Worker	Mode global	NFS commun NFS stagiaires
HDFS Namenode	Fixé	Local
HDFS Datanode	Mode global	Local
Jupyter “équipe”	Fixé	NFS commun NFS stagiaires
Jupyter “GPU”	Contraint	NFS commun NFS stagiaires
Jupyter “stagiaires”	Fixé	NFS commun (lecture seule) NFS stagiaires
PostgreSQL	Instance unique	Aucun
Proxy SOCKS5	Instance unique	Aucun

Stack 3 : Spark

```
service:
  notebook-interns:
    image: tyrex-docker.inrialpes.fr/spark-notebook
    command: ["notebook", "spark-master", "--network", "10.0.0.0/16"]
    environment:
      - TZ
    networks:
      - tls-net
    ports:
      - "8880:8880"
      - "8780:8787"
    volumes:
      - shared-notebooks:/notebooks:ro
      - shared-interns:/interns
      - ./interns_config.py:/root/.jupyter/jupyter_notebook_config.py:ro
      - /etc/localtime:/etc/localtime:ro
      - /etc/timezone:/etc/timezone:ro
    deploy:
      placement:
        constraints:
          - node.hostname == tyrex-serv5
```

Stack 3 : Spark

```
    # Proxy
    proxy:
      image: tyrex-docker.inrialpes.fr/socks5
      command: ["-p", "9050"]
      networks:
        - tls-net
      ports:
        - "9050:9050"

volumes:
  shared-notebooks:
    driver: nfs
    driver_opts:
      share: tyrex-serv4:/scratch/shared/notebooks

networks:
  tls-net:
    driver: overlay
    name: tls-net
    attachable: true
    driver_opts:
      encrypted: 1
```

Cas du serveur avec GPU – Première approche

- Serveur non associé au Swarm
- Utilisation d'une composition pour gérer le serveur :

```
version: '2.3'
# Fix to 2.3 to allow the "runtime" argument in services
services:
  # Notebook with GPU support (to execute on serv7)
  notebook-gpu:
    runtime: nvidia
    image: tyrex-docker.inrialpes.fr/anaconda-gpu
    restart: always
    environment:
      - TZ
    ports:
      - "8888:8888"
    volumes:
      - notebooks:/notebooks
      - /etc/localtime:/etc/localtime
      - /etc/timezone:/etc/timezone
# ... volume definition ...
```

Cas du serveur avec GPU – Seconde approche

- Serveur intégré au Swarm
  - Problème : Option `runtime` indisponible

Cas du serveur avec GPU – Seconde approche

- Serveur intégré au Swarm
  - Problème : Option `runtime` indisponible
  - Solution : Définir `nvidia` comme runtime par défaut
  - Avantage : Accès au GPU lors de la création d'images

Cas du serveur avec GPU – Seconde approche

- Serveur intégré au Swarm
  - Problème : Option `runtime` indisponible
  - Solution : Définir `nvidia` comme runtime par défaut
  - Avantage : Accès au GPU lors de la création d'images

- Fichier `/etc/docker/daemon.json` :

```
{
  "default-runtime": "nvidia",
  "runtimes": {
    "nvidia": {
      "path": "nvidia-container-runtime",
      "runtimeArgs": []
    }
  }
}
```

Cas du serveur avec GPU – Seconde approche

- Ajout du label au serveur (depuis un Swarm Manager) :

```
# Récupération du node ID
docker node ls | grep -i serv7
# Ajout du label GPU
docker node update --label-add gpu=1 <node-id>
```

Cas du serveur avec GPU – Seconde approche

- Ajout du label au serveur (depuis un Swarm Manager) :

```
# Récupération du node ID
docker node ls | grep -i serv7
# Ajout du label GPU
docker node update --label-add gpu=1 <node-id>
```

- Ajout à la composition du cluster :

```
version: '3.5'
services:
  # Notebook(s) with GPU support
  notebook-gpu:
    ports:
      - target: 8888
        published: 7500
        protocol: tcp
        mode: host
    deploy:
      placement:
        constraints:
          - node.labels.gpu == 1
```

État actuel du cluster

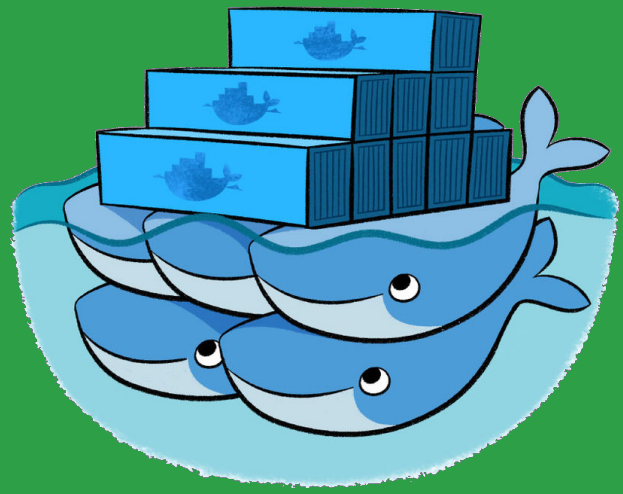
Serv2	**Serv3** *(manager)*	**Serv4**	**Serv5**
Portainer	Portainer	Portainer	Portainer
HDFS Datanode	HDFS Datanode	HDFS Datanode	HDFS Datanode
Spark Worker	Spark Worker	Spark Worker	Spark Worker
Hive	Spark Master		Notebook équipe
Spark History	Proxy SOCKS5		Notebook stagiaires

Serv6 *(leader)*	**Serv7** *(GPU)*
Portainer	Portainer
HDFS Datanode	HDFS Datanode
Spark Worker	Spark Worker
HDFS Namenode	Notebook GPU
PostgreSQL (*backend Hive*)	
Registry	

Utilisations actuelles du cluster

- Big Data Medical : analyse de données médicales américaines
  - Utilisation du cluster Spark (Spark ML)
  - Utilisation du serveur avec GPU (Tensorflow)

- muIR : optimisation de Regular Path Queries
  - Instanciation ponctuelle de divers serveurs (PostgreSQL, Virtuoso, Neo4J, ...)
  - Chargement et requêtage sur différents jeux de données (Yago2s, WatDiv, gMark, ...)



4
Retour d'expérience

Après 3 ans d'usage

Commandes usuelles

- Démarrage / mise à jour d'une *stack* :

```
$ docker stack deploy -c cluster_stack.yml spark
```

- Arrêt d'une *stack* :

```
$ docker stack rm spark
```

- Temps moyens de démarrage :
  - Avec images en cache : moins d'1 minute
  - Avec téléchargements : 5 minutes maximum (2 minutes en moyenne)

Avantages de cette approche

+ Configuration facile
+ Déploiement rapide
+ Reproductibilité
  - Exécution
  - Infrastructure
+ Redémarrage automatique et rapide des services
+ Méthodologie simple :
  - Création d'images
  - Validation sur une machine de test
  - Envoi de l'image au registre
  - Mise à jour de la *stack*

Inconvénients de cette approche

- Quasi-obligation de préparer des images personnalisées
- Pas de gestion de quotas de ressources avec Swarm
- Quelques soucis avec les montages NFS Swarm
  - Peut poser problème en cas de gros plantage Swarm
  - `umount` sur l'hôte en échec (ressource occupée)
  - Contournement possible : utiliser un point de montage local
- "Sensibilité" de Docker

“Sensibilité” de Docker – Exemple concret

- Problème :
  - Exécution d’un code chargeant 300 Go de données dans 100 Go de RAM...
  - Code exécuté sur l’hôte d’un Swarm Manager

“Sensibilité” de Docker – Exemple concret

- Problème :
  - Exécution d’un code chargeant 300 Go de données dans 100 Go de RAM...
  - Code exécuté sur l’hôte d’un Swarm Manager
- Résultat :
  1. Arrêt de processus de l’hôte par le noyau (libération de mémoire)
  2. Le Swarm Manager tombe
  3. Les autres nœuds Swarm deviennent instables

“Sensibilité” de Docker – Exemple concret

- Problème :
  - Exécution d’un code chargeant 300 Go de données dans 100 Go de RAM...
  - Code exécuté sur l’hôte d’un Swarm Manager
- Résultat :
  1. Arrêt de processus de l’hôte par le noyau (libération de mémoire)
  2. Le Swarm Manager tombe
  3. Les autres nœuds Swarm deviennent instables
- Redémarrage du cluster :

Fréquence	Opération	Durée
Toujours	Redémarrage manuel de **chaque** démon	2 min
Quelquefois	Remise à zéro de quelques démons Docker ⇒ suppression de `/var/lib/docker`	5 min
Rarement	Re-création intégrale du Swarm	1-10 min

Au sujet des rumeurs sur Docker Swarm...

> *"But its equally important for us to note that **Swarm orchestration is not going away**. Swarm forms an integral cluster management component of the Docker EE platform ; in addition, **Swarm will operate side-by-side** with Kubernetes in a Docker EE cluster, allowing customers to select, based on their needs, the most suitable orchestration tool at application deployment time."*

Source : Swarm Orchestration in Docker Enterprise Edition
(annonce officielle du support Kubernetes)

Résumé : No. Swarm is not dead. (par Bret Fisher)

Questions ?

Thomas Calmant
thomas.calmant@inria.fr

Credits :

- CommitStrip
- Laurel

SED/Tyrex
Montbonnot-Saint-Martin

Petit aperçu du cluster

CommitStrip.com