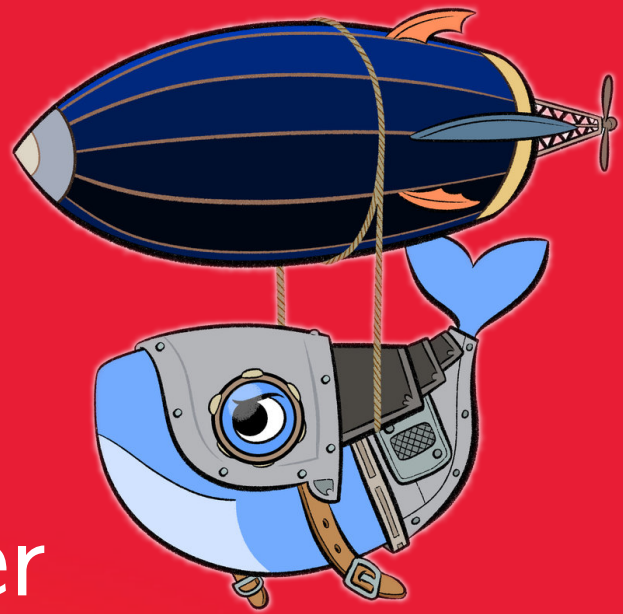
Docker @ Tyrex

1
Mise en place de Docker



Situation initiale

- Contexte :
  - Projet Datalyse : pipeline de traitement de données
  - Projet SparqlGX : requêtage sur des BDD sémantiques
- 2 machines physiques
  - 1 grappe de 6 VMs
  - 1 grappe de 12 VMs
- Problèmes
  - Lenteur des E/S
  - Consommation inutile de ressources (OS, services, . . .)
  - Contraintes de versions d'OS et de bibliothèques
- Solutions possibles : RàZ du cluster
  - Conservation de l'approche VM (OpenShift, Kubernetes, . . .)
  - Utilisation directe des hôtes
  - Approche conteneur (Docker ou rkt)

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

```
docker run --link mysql:db ...
docker run -p 8080:80 ...
    vim docker-compose.yml
    docker-compose up -d
```

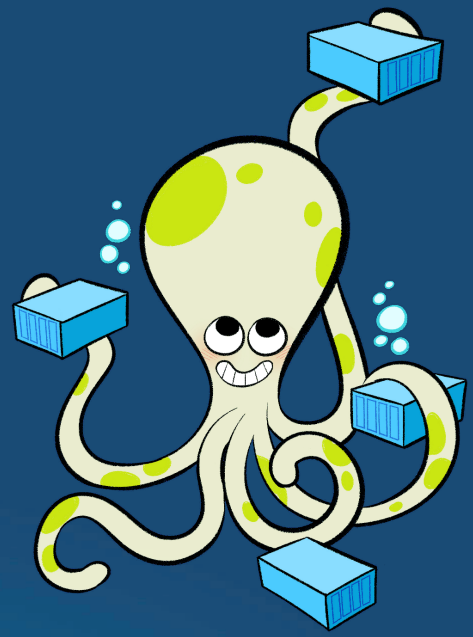
Adoption de Docker par étapes

Étape 2 : Essais entre deux serveurs

1. À la main (ligne de commande et ouverture de ports)
2. Avec Docker Swarm
   - 2.1 En ligne de commande (services)
   - 2.2 Avec les *stacks*

```
docker run -p ...
docker service create ...
docker stack deploy -c ...
```

2 Infrastructure actuelle



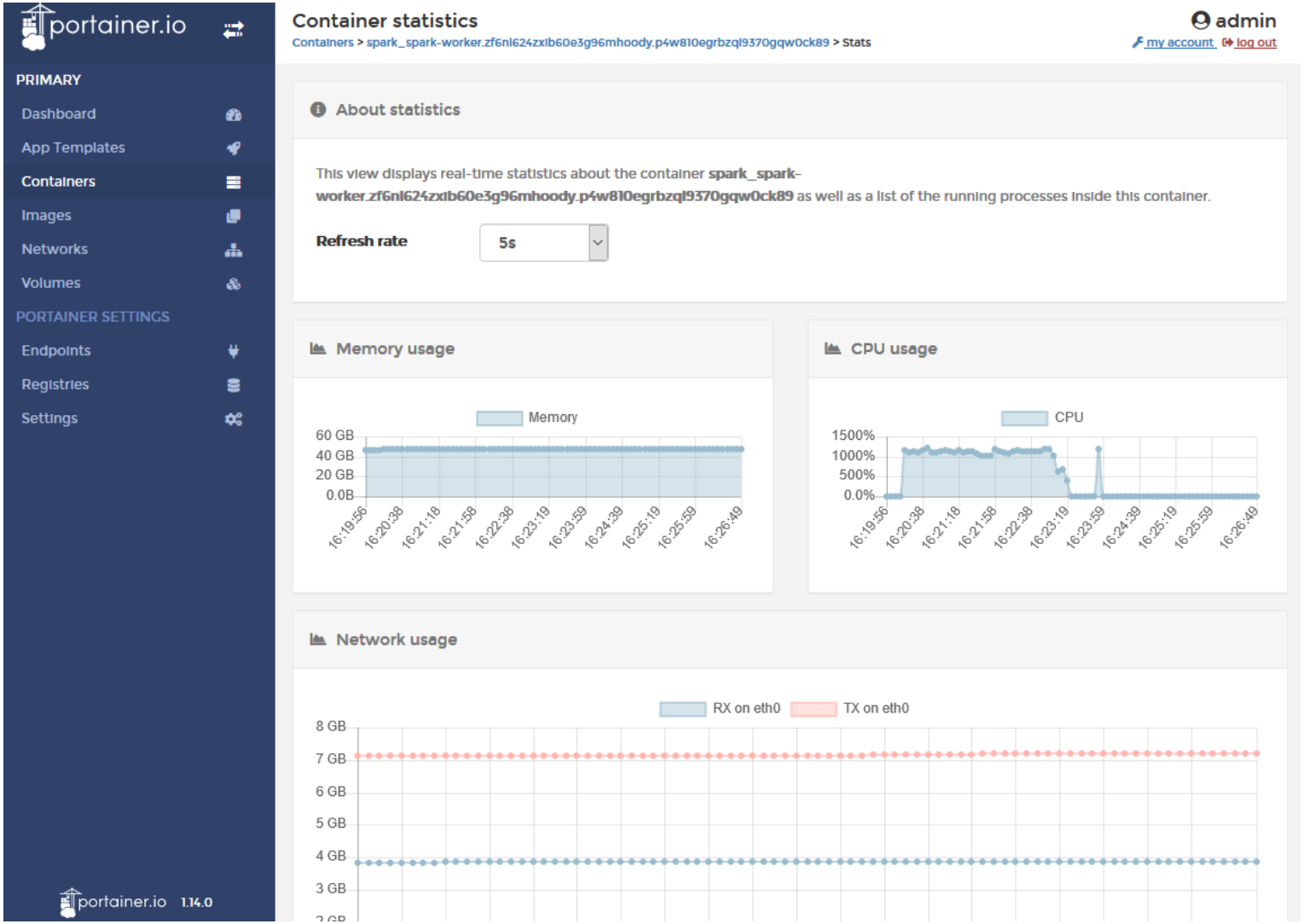
État du cluster

- 5 serveurs
  - 2 Swarm Managers
  - 1 serveur de données NFS
  - 1 réplication des données (non partagée)
- Configuration commune des *stacks*
  - Volumes NFS (avec le plugin NetShare)
  - Réseau “tls-net” (*overlay*, chiffré)
- 3 *stacks* définissent l’état standard du cluster
  D’autres *stacks* sont ajoutées selon les besoins.

Stack : 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
- Nécessite une configuration pour sécuriser son accès (mot de passe, réseau client, . . .)

Stack : Portainer



Stack : Portainer

```
version: '3.2'
services:
  portainer:
    image: portainer/portainer
    command: [-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 : Registre d'images Docker

- Stockage des images Docker personnalisées
- Accès réseau :
  - Accès par l'équipe et le cluster *via* le port 443
  - Réseau privé interne entre le frontal HTTPS et le registre

Service	Placement	Volumes
nginx	Fixé	Fichiers de configuration
registry	Fixé	Local
webui	Instance Unique	Fichier de configuration

Stack : Registre d'images Docker

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

```
  registry:
    image: registry
    networks:
      - regnet
    volumes:
      - ./data:/var/lib/registry
      - ./registry_config.yml:
        ↪  /etc/docker/registry/config.yml
    deploy:
      placement:
        constraints:
          - node.hostname == serv6

networks:
  regnet:
    driver: overlay
    driver_opts:
      internal: "true"
```

Stack : 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* proxy aux IHM de Spark et HDFS

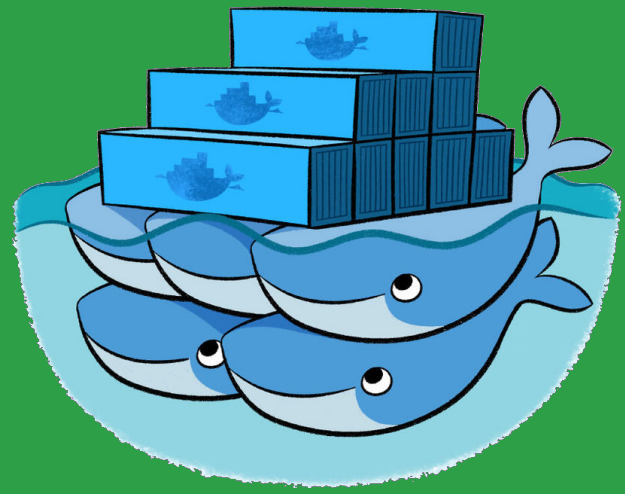
Stack : Spark

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 notebook “équipe”	Fixé	NFS commun NFS stagiaires
Jupyter notebook “stagiaires”	Fixé	NFS commun (lecture seule) NFS stagiaires
PostGreSQL	Instance unique	Aucun
Proxy SOCKS5	Instance unique	Aucun

État réel du cluster

Serv2	Serv3	Serv5	Serv6
Portainer	Portainer	Portainer	Portainer
HDFS Datanode	HDFS Datanode	HDFS Datanode	HDFS Datanode
Spark Worker	Spark Worker	Spark Worker	Spark Worker
PostGreSQL	Spark Master	Notebook équipe	nginx
Overpass API	*Elevation Server*	Notebook stagiaires	registry

- Serv4 est réservé à une expérimentation nécessitant toute la RAM disponible.
- Des serveurs de tuiles OSM et d'information cartographiques tournent pour une expérimentation d'application mobile
- Des expérimentations ponctuelles ont lieu sur Serv6



3
Retour d'expérience

Après près de 2 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

+ Déploiement rapide
+ Reproductibilité
  - Exécution
  - Infrastructure
+ Configuration facile
+ Redémarrage automatique 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 d'utiliser des images personnalisées
- Pas de quotas de ressources avec Swarm
- Quelques soucis avec Java
  (détection de la RAM totale/disponible)
- Partage de ressources avec l'hôte
  (point de montage occupé, . . .)
- "Sensibilité" de Docker

“Sensibilité” de Docker

- Problème :
  - Exécution d’un code chargeant 300 Go de données dans 100 Go de RAM...
  - Code exécuté sur l’hôte d’un Swarm Manager (pas forcément Leader)
- Résultat :
  - Arrêt de processus de l’hôte par le noyau (libération de mémoire)
  - Le Swarm Manager tombe $\Rightarrow$ les autres nœuds Swarm deviennent instables
- Conséquences :
  - Nécessité de redémarrer **chaque** démon Docker du cluster
  - Certaines fois : nécessité de **supprimer la configuration** de démons Docker
  - Rarement : re-création intégrale du Swarm

Inconvénients généraux de Docker

- Docker partage les ressources avec le système hôte
  - Problèmes de ressources occupées (points de montages, . . .)
- Pas de paternité des conteneurs
  - Impossible de tuer “tous les conteneurs d’un utilisateur”
- Pas de niveaux de droits sur l’utilisation de Docker
  - Tout le monde est “root” ou n’est rien

Questions ?

Thomas Calmant
thomas.calmant@inria.fr

Credits :

- Laurel

SED/Tyrex
Montbonnot-Saint-Martin