

Rapport projet de virtualisation

Virtualisation des SI

Rmiza, Crockford, Zeller, Guignard

Année 2017-2018

27/09/2017

Version 1.5



MESOS

Table des matières

1	Introduction	3
2	Cahier des charges	3
2.1	Partie théorique	3
2.2	Partie pratique	3
3	Partie théorique	3
3.1	Services	3
3.2	Containers	5
3.3	Hyperviseur	5
3.4	Orchestrateur	5
3.5	Docker Swarm	6
3.6	Cattle (Rancher)	8
3.7	DC/OS	9
4	Partie pratique	12
4.1	Kubernetes sur CentOS	12
4.2	Première installation de DC/OS	13
4.3	Syncthing	18
4.4	Deuxième installation de DC/OS	18
4.5	Aparté	26
5	Conclusion	26
6	Références	27
7	Table des illustrations	27

1 Introduction

Dans le cadre du cours de virtualisation des systèmes d'information, nous avons pour projet de mettre en place un système distribué, permettant à un utilisateur d'accéder à des données au moyen d'une interface utilisateur. Afin de mener ce projet à bien, nous avons partagé le travail en deux parties. La première, consistera à faire une recherche sur les différentes technologies de virtualisation. La deuxième se présentera sous la forme d'un document technique où nous décrirons la démarche, les problèmes rencontrés et les différentes configurations mises en place.

2 Cahier des charges

2.1 Partie théorique

- Étude des techniques de virtualisation appliquées sur les containers :
 - o Etienne : Swarm (Docker)
 - o Quentin : DC/OS (Mesos Apache)
 - o Thomas : Cattle (Rancher)
- Comparaison des différents orchestrateurs et hyperviseurs disponibles.
- Recherche des services à déployer sur l'infrastructure virtualisée (partage NFS, serveur web)

2.2 Partie pratique

- Choix de l'orchestrateur et l'hyperviseur.
- Choix du service
- Installation et configuration de l'infrastructure virtualisée
- Mise en place des services sur l'infrastructure virtualisée

3 Partie théorique

3.1 Services

L'objectif principal du projet est de mettre en place et administrer un cluster sur lequel sera déployé un ou plusieurs services containerisés. Les services ayant un intérêt pour ce projet de déploiement de containers sont ceux proposant des applications liées aux systèmes de fichiers ainsi qu'à leur manipulation à travers un réseau de partage. Suite au travail de recherche effectué pour trouver un service fiable avec une version containerisée stable, trois services se sont démarqués : Syncthing, Resilio et Nextcloud.

3.1.1 Syncthing

Syncthing est une application multiplateforme de synchronisation de fichiers pair à pair open source. Contrairement à d'autres services similaires, aucun enregistrement préalable n'est requis, ce qui est intéressant dans notre cas. Syncthing met en œuvre son propre BEP open source que nous avons du reste étudié dans le cours de protocole réseau dispensé par Monsieur Heardt.

Syncthing est construit selon le modèle suivant : les utilisateurs fournissent le matériel sur lequel travaille le logiciel. Par conséquent, plus le nombre de nœuds est important, plus grande sera l'efficacité des transferts si on fournit assez de bande passante. Il prend en charge IPv6 et le relayage pour IPv4 pour permettre à deux nœuds se trouvant chacun derrière un pare-feu ou passerelle NAT (tels un routeur d'entreprise ou une box Internet) de communiquer via un serveur relais. Le relayage est de même nature que celui du protocole TURN, c'est-à-dire avec un chiffrement TLS de bout en bout entre les nœuds. Ainsi, les relais ne peuvent pas voir les données mais seulement le flux chiffré et par conséquent, les adresses

IP source et destination en clair. On peut aussi avoir son propre serveur relais, public ou privé, et s'en servir exclusivement ou en collaboration avec les autres relais publics.

3.1.2 Resilio Sync

Resilio Sync, anciennement BitTorrent Sync, est un outil multiplateforme de synchronisation de fichiers pair à pair. Il peut synchroniser des fichiers entre des périphériques sur un réseau local ou entre des périphériques distants via Internet. Il apporte une réponse à de nombreuses préoccupations concernant les limites de stockage de fichiers, la confidentialité, les coûts ou encore les performances gérés par les services existants.

La synchronisation des fichiers repose sur le protocole BitTorrent ce qui assure un résultat optimal au vu de la fiabilité de ce protocole. Les données de l'utilisateur sont stockées sur le périphérique local de l'utilisateur plutôt que dans un cloud. Cela nécessite donc au moins deux nœuds utilisateurs pour être en ligne afin de synchroniser les fichiers entre eux. Resilio Sync chiffre les données au moyen d'une clé AES-128 qui peut être générée aléatoirement ou définie par l'utilisateur. Cette clé est dérivée d'un "secret" qui peut être partagé avec d'autres utilisateurs pour partager des données. Les données sont ensuite envoyées directement entre les périphériques, à moins que le périphérique cible ne soit inaccessible (par exemple derrière un pare-feu) auquel cas les données seront d'abord relayées via un nœud intermédiaire. Dans une topologie de maillage réseau, de nombreux périphériques peuvent être connectés simultanément. De même, plusieurs fichiers peuvent être partagés entre eux. Il n'y a aucune limite sur la quantité de données pouvant être synchronisées si ce n'est, l'espace libre disponible sur chaque périphérique.

3.1.3 Nextcloud

Nextcloud est un service d'hébergement de fichiers. Au niveau des fonctionnalités, il est similaire à Dropbox, bien que Nextcloud soit open-source et par conséquent permette à quiconque de l'installer et de l'utiliser sur un serveur privé.

Contrairement aux services propriétaires comme Dropbox, l'architecture ouverte de Nextcloud permet à l'utilisateur d'ajouter des fonctionnalités supplémentaires au serveur sous la forme d'applications mais aussi d'avoir un contrôle total de ses données. Les fichiers Nextcloud sont stockés dans des structures de répertoires conventionnelles et peuvent être consultés par exemple via WebDAV. Les fichiers utilisateurs sont chiffrés pendant le transit avec HTTPS.

Du point de vue de l'administration, Nextcloud permet de gérer des utilisateurs et des groupes (via OpenID ou LDAP). Le contenu peut être partagé en définissant des autorisations en lecture et écriture granulaires entre les utilisateurs et les groupes. Alternativement, les utilisateurs de Nextcloud peuvent créer des URL publiques lors du partage de fichiers.

3.2 Containers

Une virtualisation matérielle s'appuie sur ce qu'on appelle un hyperviseur. Celui-ci se pose sur le matériel de l'hébergeur qui le répartit proportionnellement aux systèmes hôtes. A l'inverse, aucun système d'exploitation supplémentaire n'est démarré lors de la virtualisation par conteneur. Au lieu de cela, l'OS commun forme des instances isolées. Ces conteneurs virtuels fournissent aux applications un environnement complet d'exécution.

Les conteneurs logiciels sont considérés comme des applications pour serveur. Pour installer une application, il faut charger le conteneur correspondant dans un format portable (une image) avec toutes les données nécessaires sur l'ordinateur puis, le démarrer dans un environnement virtualisé. On peut implémenter les applications dans les conteneurs avec presque tous les systèmes d'exploitation.

3.3 Hyperviseur

Un hyperviseur, également appelé gestionnaire de machine virtuelle, est un programme qui permet à plusieurs systèmes d'exploitation de partager un seul hôte matériel. En apparence, chaque système d'exploitation a l'exclusivité du processeur, de la mémoire et de toutes les autres ressources de l'hôte. Mais en réalité, c'est l'hyperviseur qui contrôle le processeur et les ressources de l'hôte, allouant alternativement à chaque système d'exploitation ce dont il a besoin et s'assurant que ces systèmes invités (ou machines virtuelles) n'interfèrent pas l'un avec l'autre.

3.4 Orchestrateur

L'orchestration est un processus qui consiste à gérer de manière intelligente et autonome un système complexe via la surveillance de services ou au niveau des ressources (CPU, RAM, disque), dans le cas d'un cluster de machines virtuelles. Il peut aussi prendre des décisions en cas de problème ou de comportement anormal. Voici les principaux orchestrateurs qui existent sur le marché :

- Kubernetes
- Swarm
- Rancher
- Mesos

3.5 Docker Swarm

Docker Swarm est un outil de clusterisation et d'orchestration utilisé pour manager des conteneurs sur plusieurs nœuds. Comme on peut le voir sur la figure ci-dessous, l'architecture Docker Swarm se compose de *manager* et de *worker*. L'utilisateur peut exécuter divers services dans le cluster Swarm à l'aide de fichiers de configuration au format YAML.

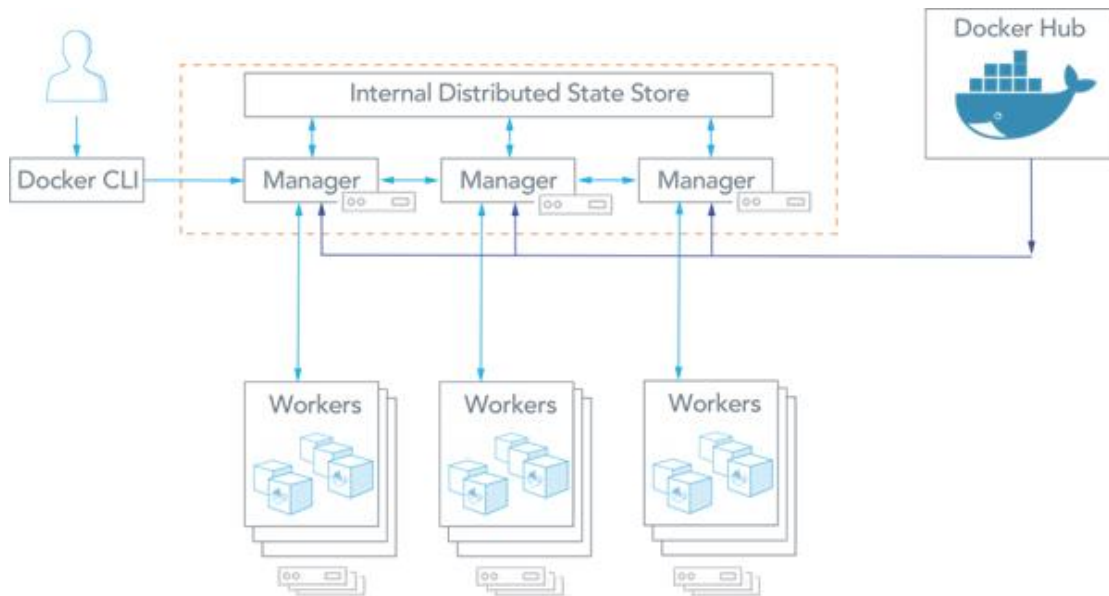


Figure 1: Architecture Docker

Voici quelques termes communs associés à Docker Swarm :

- Un nœud est une instance de Swarm. Les nœuds peuvent être distribués sur place ou dans des infrastructures virtualisées publiques (cloud).
- Swarm est un cluster de nœuds ou *Docker Engines*.
- Les *worker node* sont des nœuds qui collectent et exécutent des tâches à partir du *manager node*.
- Les *manager node* reçoivent des directives de service de l'utilisateur et répartissent les travaux sur les *worker node*. Les *manager node* peuvent également remplir les tâches des nœuds travailleurs.
- Un service spécifie l'image à contenir et le nombre de répliques.
- Une tâche est un service programmé sur un *worker node*

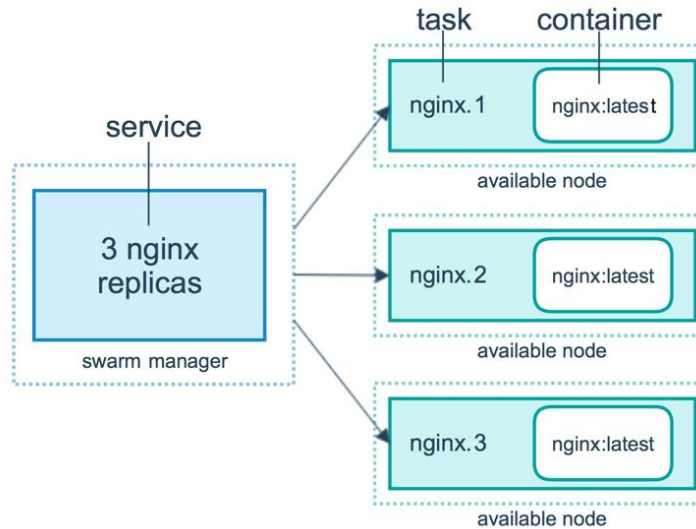


Figure 2: Cluster Docker

3.5.1 Évolutivité

Docker Swarm est capable de déployer des conteneurs très rapidement, même dans des clusters très volumineux et avec des niveaux de remplissage élevés. Cela permet notamment d'avoir des temps de réaction rapides en fonction de la demande. D'autre part, les nouvelles répliques peuvent être démarrées en une seule ligne de commande.

3.5.2 Haute disponibilité

Docker Swarm assure la haute disponibilité des services grâce à la réplication. Le même conteneur est déployé sur plusieurs nœuds pour ajouter une redondance et se redéployer à nouveau si un hôte exécutant le service n'est plus disponible. Bien que l'orchestrateur de conteneur peut être exécuté sur un seul serveur, des nœuds supplémentaires sont nécessaires pour une redondance réelle.

3.5.3 Équilibrage de charge

Docker Swarm offre un équilibrage de charge intégré. Tous les conteneurs d'un cluster unique rejoignent un réseau commun assurant la connexion de n'importe quel nœud à n'importe quel conteneur. Les requêtes de connexion via n'importe quel nœud dans Swarm sont redirigées en interne vers un nœud exécutant une instance du service. Swarm comporte un DNS qui peut être utilisé pour distribuer des requêtes entrantes à un nom de service. Les services peuvent être exécutés sur des ports spécifiés par l'utilisateur ou peuvent être affectés automatiquement.

3.5.4 Accessibilité des services

Swarm manager attribue à chaque service un nom DNS, une adresse IP virtuelle privée et un nom de service, ce qui simplifie grandement la découverte du service.

3.5.5 Volume de stockage

Les volumes de données de Docker sont des répertoires partagés dans un ou plusieurs conteneurs. Les volumes sont créés séparément, ou avec des conteneurs, et peuvent être partagés entre plusieurs conteneurs. Les volumes de données persistent même lorsque les conteneurs qui les utilisent sont supprimés. Les volumes par eux-mêmes ne sont cependant que locaux sur le nœud sur lequel ils sont créés. Pour créer des volumes globaux, le moteur Docker prend en charge les plugins de volume.

3.5.6 Réseau

Docker Swarm forme une superposition de réseau multi-hôte qui relie les conteneurs fonctionnant sur tous les nœuds du cluster. Il est possible de configurer manuellement des réseaux inter-conteneurs. Les connexions entre les nœuds sont automatiquement sécurisées via l'authentification TLS avec des certificats.

3.5.7 Performances

D'après¹ l'article suivant "Docker's blog post on scaling Swarm clusters", Docker Swarm été testé avec jusqu'à 30 000 conteneurs et 1 000 nœuds avec 1 *Swarm manager*.

3.6 Cattle (Rancher)

Rancher² est une plateforme open source permettant de gérer l'exécution de conteneurs. Il intègre l'orchestration de l'infrastructure en contrôlant un ensemble d'hôtes Linux, qu'ils soient dans un cloud, privé ou public, ou qu'ils s'agissent de machines virtuelles ou physiques. Il dispose d'une interface graphique pour effectuer des actions mais aussi pour visualiser rapidement l'état de l'infrastructure et des applications. Rancher inclue une distribution de tous les orchestrateurs les plus populaires tels que Docker Swarm, Kubernetes et Mesos. L'utilisateur peut donc choisir, avec Rancher, de créer de multiples clusters Swarm ou Kubernetes pour ensuite monitorer leurs applications. En plus de ces orchestrateurs, Rancher offre son propre orchestrateur nommé Cattle. Cet orchestrateur est utilisé extensivement par Rancher pour orchestrer, mettre en place et monitorer, mais aussi dans le but d'optimiser des clusters Swarm, Kubernetes et Mesos.

3.6.1 Orchestration de l'infrastructure

Rancher supporte des données brutes provenant de tout cloud public ou privé sous la forme d'hôtes Linux. Du point de vue de Rancher, une VM instanciée depuis un cloud ou un serveur sont bien distinctifs. Rancher implémente une couche portable de services d'infrastructure, développés spécialement pour déployer des applications containerisées. Ces services incluent la partie réseau, le stockage, le répartiteur de charge, le DNS et la sécurité. Ces services sont typiquement déployés sous la forme de containers pour permettre à Rancher de faire tourner la même infrastructure sur n'importe quel hôte Linux et sur n'importe quel cloud.

3.6.2 Configuration

Cattle³ est basé sur des commandes Docker et le déploiement d'applications est organisé dans des *stacks*, qui peuvent être lancés directement depuis un catalogue d'applications fourni depuis un fichier *docker-compose.yml* et *rancher-compose.yml*, ou créé directement depuis l'IU. Chaque stack est composé de services qui sont principalement une image Docker. Il est possible d'inclure des services de répartition de charge et d'autres services externes au sein d'un stack Cattle.

¹ <https://blog.docker.com/2015/11/scale-testing-docker-swarm-30000-containers/>

² <http://rancher.com/docs/rancher/latest/en/>

³ <http://rancher.com/cattle-swarm-kubernetes-side-side/>

3.7 DC/OS

DC/OS (Datacenter Operating System) est un système d'exploitation distribué et open-source basé sur le kernel Apache Mesos. Il administre de multiples machines sur un cloud ou sur place à partir d'une interface unique. Il déploie des containers, systèmes distribués et autres applications sur ces machines. En plus, il apporte la découverte de service et le management de ressources pour faire tourner ces services et les faire communiquer les uns avec les autres.

En tant que plateforme, DC/OS se distingue de la couche infrastructure dans le sens où son infrastructure peut être constituée d'hardware physique ou virtuel.

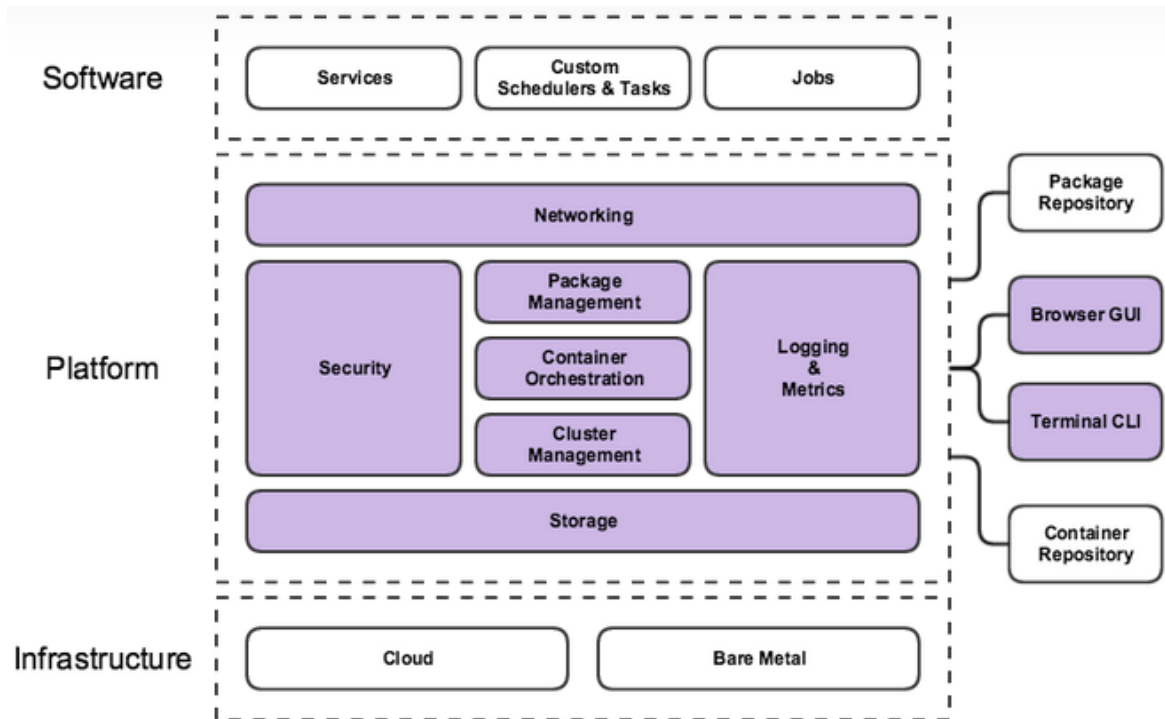


Figure 3: Architecture de DC/OS

La couche software permet à DC/OS de gérer et d'installer plusieurs types de services : bases de données, files d'attente de messages, solutions de surveillance, gestion des contrôles de source, etc. En plus de ces applications et services packagés, l'utilisateur peut installer ses propres applications, services et tâches planifiées. Au niveau de la couche plate-forme, des dizaines de composants sont disponibles dans les catégories suivantes :

- Gestion de cluster
- Orchestration de conteneur
- Runtime des conteneurs
- Log et métriques
- Réseaux
- Gestion des paquets
- IAM et sécurité
- Stockage

Ces composants sont répartis entre plusieurs types de nœuds :

- Les masters
- Les agents privés
- Les agents publics

Pour que DC/OS soit installé, chaque nœud doit déjà être approvisionné avec l'un des systèmes d'exploitation supportés. Au niveau de la couche d'infrastructure, DC/OS peut être installé sur des serveurs publics, des serveurs privés ou du matériel physique. Certaines de ces cibles d'installation ont des outils de provisionnement automatisés mais presque n'importe quelle infrastructure peut être utilisée, à condition qu'elle inclue plusieurs machines x86 sur un réseau IPv4 partagé. L'espace du noyau DC/OS est composé de masters Mesos et d'agents Mesos. L'espace utilisateur comprend des composants système tels que Mesos-DNS, Distributed DNS Proxy mais également, des services tels que Marathon ou Spark. L'espace utilisateur comprend aussi des processus gérés par les services par exemple, une application Marathon.

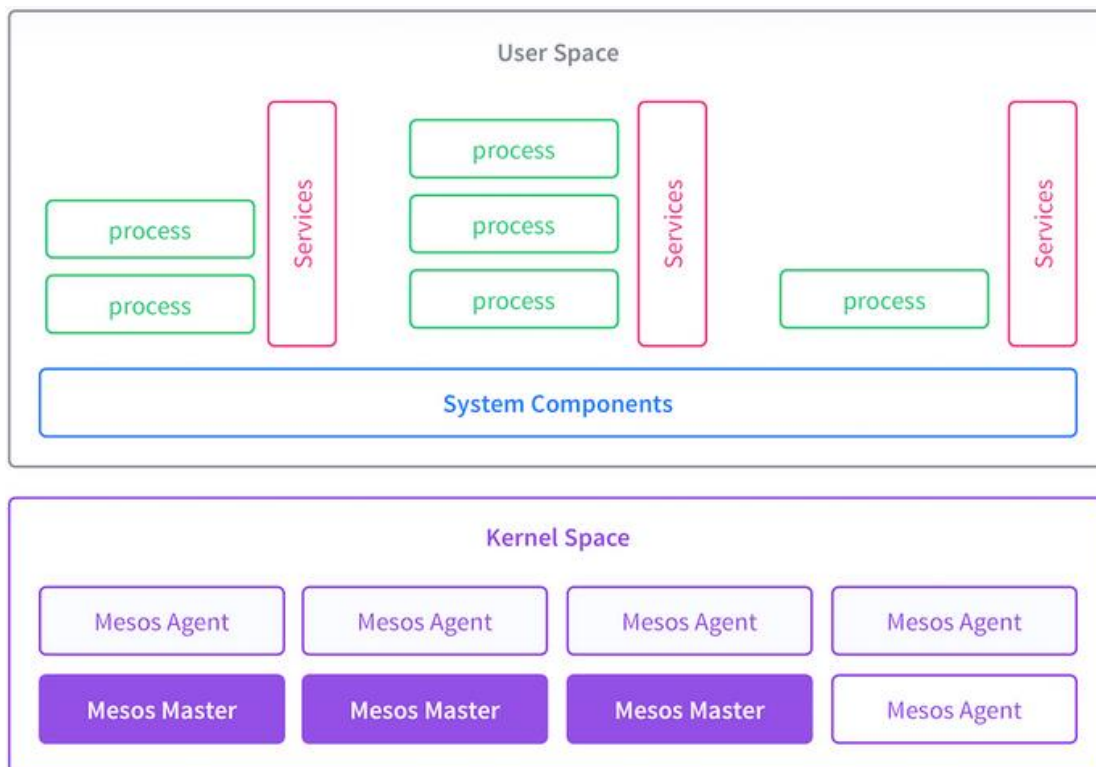


Figure 4: Espace DC/OS

3.7.1 Espace Kernel

Le *kernel Space* gère l'allocation des ressources et la planification dans le cluster à deux niveaux. En effet, l'espace kernel contient deux types de processus, les masters et agents Mesos :

- Mesos masters : Le processus mesos-master orchestre les tâches exécutées sur les agents Mesos. Le processus Mesos Master reçoit les rapports de ressources des agents Mesos et les distribue aux services DC/OS enregistrés, tels que Marathon ou Spark. Lorsqu'un master Mesos principal échoue en raison d'une panne ou se déconnecte pour une mise à niveau, un master Mesos en attente devient automatiquement le leader sans perturber les services en cours d'exécution.
- Mesos agents : les nœuds de l'agent Mesos exécutent des tâches Mesos discrètes pour le compte d'un framework. Les nœuds d'agents privés exécutent les applications et les services déployés via un réseau non routable. Les nœuds d'agent public exécutent des applications et des services DC/OS dans un réseau accessible au public. Le processus mesos-slave sur un agent Mesos gère ses ressources locales (cœurs CPU, RAM, etc.) et enregistre ces ressources auprès des maîtres Mesos. Il accepte également les demandes de planification du master Mesos et appelle un Executor pour lancer une tâche via des containers :
 - o Mesos fournit une conteneurisation légère et l'isolation des ressources des exécuteurs en utilisant des fonctionnalités spécifiques à Linux telles que les groupes de contrôle et les espaces de noms.
 - o Docker prend en charge le lancement de tâches contenant des images Docker.

3.7.2 Espace utilisateur

Le *user space* couvre les composants système et les services tels que Chronos ou Kafka. Les composants système sont installés et s'exécutent par défaut dans le cluster et comprennent les éléments suivants :

- Le routeur Admin est une configuration open source NGINX qui fournit une authentification centrale et un proxy aux services DC/OS.
- L'exposant configure automatiquement ZooKeeper pendant l'installation et fournit une interface utilisateur Web utilisable à ZooKeeper.
- Mesos-DNS permet la découverte de services, ce qui permet aux applications et aux services de se trouver en utilisant le système de noms de domaine (DNS).
- Minuteman est l'équilibreur de charge de la couche interne 4.
- Distributed DNS Proxy est le répartiteur DNS interne.
- Marathon, l'instance native de Marathon qui est le système d'initialisation pour DC/OS, démarre et surveille les services DC/OS.
- ZooKeeper, un service de coordination performant qui gère les services DC/OS.
- Services :
 - o Un service dans DC/OS se compose d'un planificateur (responsable de la planification des tâches pour le compte d'un utilisateur) et d'un exécuteur (exécution des tâches sur les agents).
 - o Des applications au niveau de l'utilisateur, par exemple un serveur Web NGINX lancé par Marathon.

4 Partie pratique

4.1 Kubernetes sur CentOS

Nous⁴ avons fait une première tentative d'installation du Kubernetes sur CentOS. Cependant, étant donné qu'un autre groupe avait déjà choisi cette option, nous avons dû l'abandonner. Voici quand même les configurations que nous avons effectuées.

4.1.1 Configuration du master

Pour l'installation et la configuration du master, nous avons effectué les étapes suivantes :

- Installer NTPD sur le nœud master qui est un serveur de synchronisation de temps.
- Désactiver firewall et iptables qui sont activés de base sur les distributions RHEL.
- Installer ETCD et Kubernetes
 - o ETCD : est une base de données permettant l'échange de données dans un cluster CentOS. Il tolère les défaillances même quand il s'agit du nœud maître et permet de refaire une élection.
 - o Kubernetes : l'orchestrateur, grâce aux dépendances, permet d'installer tous les paquets nécessaires, dont docker évidemment.
- Éditer la base de données ETCD pour que les ports d'écoute dans */etc/etcd/etcd.conf* soient bien ouverts sur le port 2379.
- Éditer la base de données Kubernetes : s'assurer que l'écoute de l'api server ne se fasse pas uniquement en localhost. Nous pouvons aussi y assigner les adresses privées et masques des services futures.
- Activer les services : ETCD kube-apiserver kube-controller-manager kube-scheduler.
- Flannel est un utilitaire de réseau permettant de passer outre le fait que nous ne possédons pas un /24 par machine physique. Il va donc s'occuper de faire les réseaux entre les nœuds ainsi que de passer à travers les firewalls si nécessaire. Il s'occupera aussi de la résolution des noms. Il y a d'autres services de ce type mais celui-ci est couramment utilisé avec Kubernetes-CoreOS. À l'origine, il a d'ailleurs été développé par ce dernier.
 - o Définir dans la ETCD : `etcdctl mk /atomic.io/network/config => {"Network" : "172.17.0.0/16"}`

4.1.2 Configuration des minions

Pour l'installation et configuration des minions, nous avons effectué les étapes suivantes :

- Installer Flannel et Kubernetes
- Binder ETCD : */etc/sysconfig/flanneld*
- Binder Kubernetes : */etc/kubernetes/config*
- Configurer le service kubelet avec l'api master et sa propre adresse.
- Répéter les actions pour les autres minions.
- Lancer finalement les services : *kube-proxy kubelet docker flanneld*

⁴ <https://severalnines.com/blog/installing-kubernetes-cluster-minions-centos7-manage-pods-services>

4.1.3 Installation tableau de bord

Pour l'installation du tableau de bord, nous avons effectué les étapes suivantes :

- Création du fichier de configuration : `kubectrl create -f`
- Copie du fichier YAML⁵ pour la configuration :
- À noter, il est possible d'y accéder par le master mais également par les minions en les connectant à l'api. Il est ensuite possible de manager par le port 8081.

4.2 Première installation de DC/OS

L'objectif de ce système est de mettre en place, avec les trois machines physiques à disposition tournant chacun sur une machine virtuelle CentOS, un cluster comportant des agents et un master. Tout au long de cette partie, nous allons décrire les diverses étapes de l'installation et de la configuration nécessaires à la mise en place de ce type d'infrastructure. Afin de bien comprendre notre démarche quant aux différents composants physiques et virtuels, voici le schéma réseau :

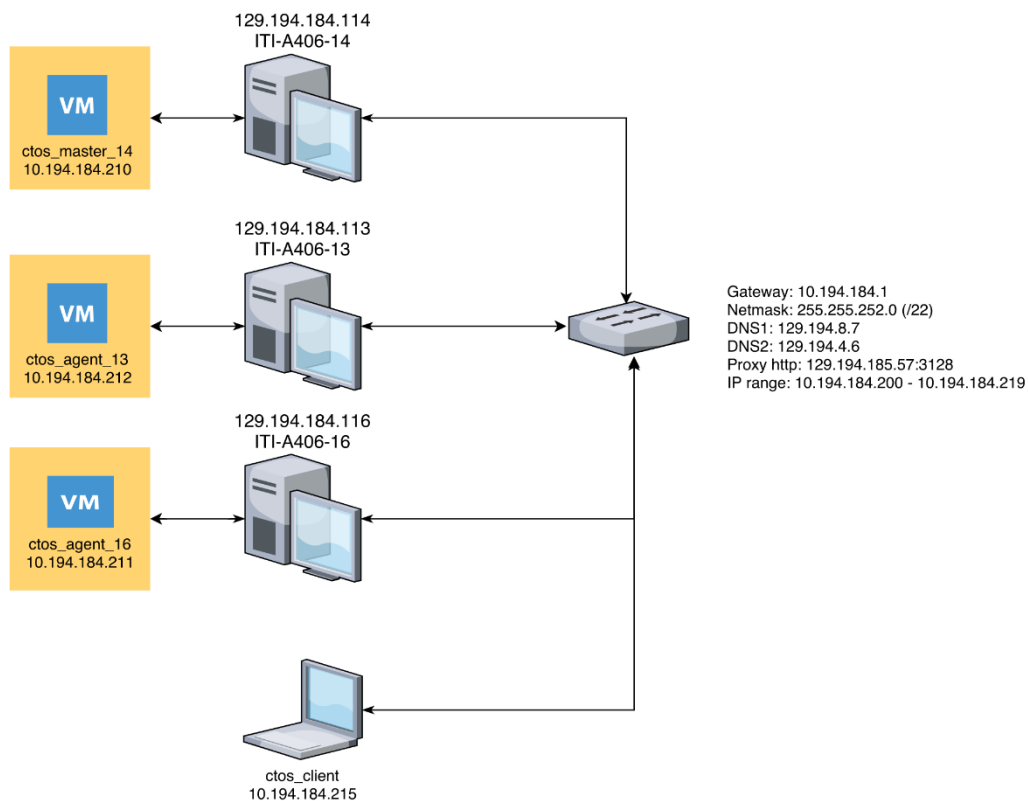


Figure 5: Plan réseau partie 1

⁵ <https://raw.githubusercontent.com/kubernetes/dashboard/master/src/deploy/recommended/kubernetes-dashboard.yaml>

4.2.1 CentOS

Les machines virtuelles ont été configurées avec un ISO minimal de CentOS-7. CentOS a été installé sur le masters et ses agents, en configurant leur hostname, IP, DNS, gateway, proxy et leur netmask. En prérequis, il a été nécessaire d'installer les commandes nano et wget sur les machines virtuelles CentOS pour pouvoir effectuer des modifications sur les fichiers de configuration. Plusieurs composants tels que NTP et Unzip ont été installé :

```
ntptime
adjtimex -p
timedatectl
sudo yum install -y tar xz unzip curl ipset
```

Ensuite, il a fallu arrêter le firewall afin d'assurer la disponibilité de tous les services lors de l'installation.

```
sudo systemctl stop firewalld
sudo systemctl disable firewalld
```

Sur chaque machine virtuelle, nous avons activé SELinux en mode *permissive* :

```
sudo sed -i s/SELINUX=enforcing/SELINUX=permissive/g /etc/selinux/config &&
sudo groupadd nogroup &&
sudo groupadd docker &&
sudo reboot
```

Les PC clients ont été configurés pour permettre de communiquer à l'extérieur du réseau en passant par un proxy. Cette tâche a été effectuée en modifiant les fichiers de la manière suivante :

- Paramètres des variables d'environnement :

```
sudo nano /etc/profile

MY_PROXY="http://129.194.185.57:3128/"
HTTP_PROXY=$MY_PROXY
HTTPS_PROXY=$MY_PROXY
FTP_PROXY=$MY_PROXY
http_proxy=$MY_PROXY
https_proxy=$MY_PROXY
ftp_proxy=$MY_PROXY
export HTTP_PROXY HTTPS_PROXY FTP_PROXY http_proxy https_proxy ftp_proxy
```

- Config du fichier yum.conf :

```
sudo nano /etc/yum.conf

proxy=http://129.194.185.57:3128/
```

- Config de wgetrc :

```
sudo nano /etc/wgetrc

http_proxy = http://129.194.185.57:3128/
https_proxy = http://129.194.185.57:3128/
ftp_proxy = http://129.194.185.57:3128/
use_prox = on
```

4.2.2 Clés SSH

Génération de la clé privée SSH via la commande suivante (clé privée : pas de mot de passe) :

```
ssh-keygen -t rsa -b 4096 -C "mesos@example.com" -f $HOME/.ssh id_rsa_mesos
```

Ajout de la clé publique dans la liste des clés autorisées :

```
mkdir -p ~/.ssh (SSH à l'hôte distant)
scp -P 22 id_rsa_mesos.pub root@<ip>:/root/.ssh
cat /root/.ssh/id_rsa_mesos.pub >> /root/.ssh/authorized_keys (clé publique copiée dans ~/.ssh/authorized_keys)
```

Une fois l'installation et la configuration terminées, les machines virtuelles étaient prêtes à recevoir les installations de Docker et DC/OS.

4.2.3 Installation de Docker

À ce stade, Docker a été installé sur tous les nœuds car CentOS ne possède pas Docker nativement. Dans la documentation, ils recommandent en effet d'installer Docker en configurant ses référentiels et en l'installant depuis ces derniers pour une installation simplifiée et des tâches mises à jour.

Installation des packages requis : *yum-utils* fournit l'utilitaire *yum-config-manager*, et *device-mapper-persistent-data* et *lvm2* sont requis par le pilote de stockage *devicemapper*.

```
sudo yum install -y yum-utils \
device-mapper-persistent-data \
lvm2
```

- La commande suivante a été utilisée pour configurer un référentiel stable :

```
sudo yum-config-manager \
--add-repo \
https://download.docker.com/linux/centos/docker-ce.repo
```

- Commande pour installer la dernière version de Docker CE :

```
sudo yum install docker-ce
```

- Démarrage de Docker et vérification si l'installation est réussie en lançant l'image hello-world :

```
sudo systemctl start docker
sudo docker run hello-world
```

4.2.4 Installation de DC/OS

Avec Docker installé, l'étape suivante a été d'installer DC/OS. Nous avons téléchargé le script d'installation sur leur site puis, nous l'avons exécuté.

```
mkdir -p /tmp/dcos-install  
cd /tmp/dcos-install  
wget https://downloads.dcos.io/dcos/EarlyAccess/dcos_generate_config.sh  
bash dcos_generate_config.sh --web
```

Voici un exemple de l'interface web présentée après l'exécution du script :

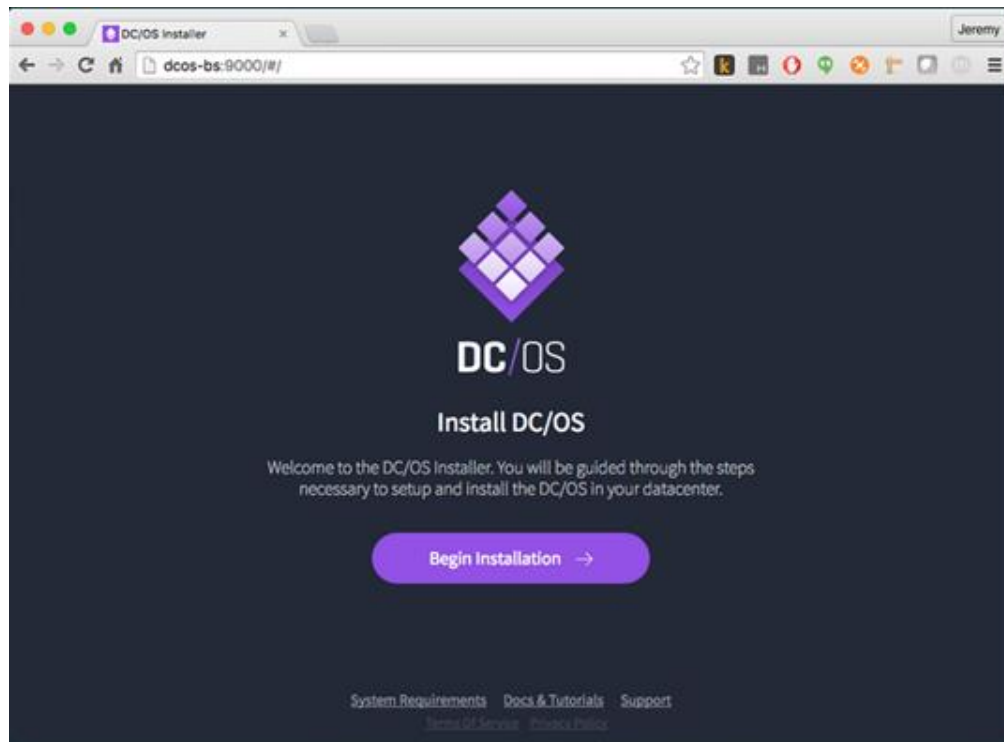


Figure 6: Web UI DC/OS

Il s'en suit plusieurs étapes de configuration telles que l'IP des agents ou l'ajout de la clé SSH privée pour la communication entre les agents et le master. Une fois cela fait, il y a trois étapes de vérifications : *pre-flight*, *post-flight* et *deploy*. À chacune de ces étapes, nous avons rencontré des erreurs qui seront présentées au point suivant.

4.2.5 Problèmes rencontrés

Tout au long de l'installation, nous avons rencontré plusieurs problèmes majeurs et mineurs. Les problèmes majeurs ont été documentés dans le repo Git que nous avons créé pour l'occasion. Voici ces différents problèmes :

- Nous avons exécuté le script d'installation sur la machine qui nous servait de master. Cependant, un des ports utilisés par le script était nécessaire au déploiement sur les agents. Nous avons résolu le problème en exécutant le script sur une autre machine.
- Entre chaque tentative de déploiement, la configuration était sauvegardée dans des fichiers de configuration. Or, les fichiers de la configuration précédente étant chargés à chaque nouvelle tentative d'installation, certaines modifications n'étaient pas prises en compte. Pour remédier à ce problème, il a fallu supprimer tous les fichiers de configuration à chaque nouvelle tentative d'installation.
- Les permissions SELinux, mises en mode *permissive* comme conseillé dans la documentation, ont dû être totalement désactivées.

```
nano /etc/sysconfig/selinux
```

```
SELINUX=disabled
SELINUXTYPE=targeted
SETLOCALDEFS=0
```

Voici un extrait du dernier message d'erreur auquel nous avons été confrontés, il spécifiait que les nœuds n'étaient pas correctement synchronisés alors que NTP avait bien été configuré :

```
{ "status": 2, "checks": { "clock_sync": { "output": "Clock is out of sync / in unsync state. Must be synchronized for proper operation.\n", "status": 2 }, "components_agent": { "output": "time=\"2017-10-31T15:10:43+01:00\" level=fatal msg=\"Error executing DC/OS components health check: exit status 7\"\n", "status": 1 }, "curl": { "output": "", "status": 0 }, "ip_detect_script": { "output": "", "status": 0 }, "journal_dir_permissions": { "output": "directory /run/log/journal has the group owner `systemd-journal` and group permissions r-x\n", "status": 0 }, "mesos_agent_registered_with_masters": { "output": "time=\"2017-10-31T15:10:43+01:00\" level=fatal msg=\"Error executing DC/OS metrics snapshot check: Unable to get url: exit status 7\"\n", "status": 1 }, "tar": { "output": "", "status": 0 }, "unzip": { "output": "", "status": 0 }, "xz": { "output": "", "status": 0 } } }
Connection to 10.194.184.212 closed.
10.194.184.212:22
```

Figure 7: Exemple d'erreur sur DC/OS

Après avoir cherché durant plusieurs cours comment résoudre tous ces problèmes, nous n'avons cependant pas réussi à comprendre pourquoi l'environnement clustérisé créé ne fonctionnait toujours pas lors des tentatives de déploiement. Nous avons alors essayé de refaire toute la configuration en ligne de commande car dans certains cas, on remarque que l'utilisation de la GUI pose des problèmes, mais sans succès. Le délai de développement du projet nous a contraint à abandonner cette solution et à faire la mise en place du cluster sur une seule machine physique avec plusieurs machines virtuelles supportant une configuration DC/OS.

4.3 Syncthing

Le premier service choisi pour ce projet a été Syncthing : une application de synchronisation simple et intuitive dans son utilisation mais aussi intéressante dans son déploiement.

4.3.1 Implémentation du container dans DC/OS

Dans un souci de compréhension et de prise en main du logiciel, Syncthing a tout d'abord été installé sur un PC et sur un smartphone. Ensuite, un essai de partage de fichiers a été effectué entre les deux appareils. Pour ce faire, le QR code donné par le logiciel à l'ordinateur portable a été entré dans le logiciel installé sur le smartphone. Cette étape a servi de synchronisation entre les deux appareils suite à laquelle, le partage s'est fait avec succès et sans problèmes majeurs. La deuxième étape a été de récupérer une image docker de Syncthing et de l'implémenter dans DC/OS afin de le déployer dans notre cluster.

4.3.2 Problèmes rencontrés

L'image containerisée de Syncthing a dû être déployée sur chaque poste du cluster utilisant docker. Une configuration existait sur internet mais elle ne permettait pas d'accéder à l'ensemble des fonctionnalités de Syncthing. Le fichier de configuration prenait comme paramètres d'utilisation deux ports : 8384 et 22000. Suite à l'installation de docker et du déploiement de Syncthing sur une machine, une erreur est apparue au moment du lancement : il s'agissait d'un problème de port qui était déjà utilisé.

Le déploiement de Syncthing au sein du cluster n'a pas pu être réalisé à cause du fonctionnement même de l'application. Ce problème était lié au QR code propre à Syncthing, nécessaire à l'identification de chaque nœud. Tous les nœuds possèdent un QR code unique et chaque nœud doit connaître les QR codes de tous les nœuds disponibles. Par conséquent, c'est la complexité dans le partage du QR code au sein du cluster et la mise en place d'une base de données pour récupérer chaque QR code qui a contraint l'abandon de la solution Syncthing au sein du projet.

4.4 Deuxième installation de DC/OS

Après avoir passé plusieurs heures sur la réalisation de la première installation de DC/OS sur les trois machines physiques, qui s'est du reste soldée par un échec, nous avons décidé de réitérer l'installation mais cette fois-ci, en utilisant une seule machine physique. Cela a impliqué l'installation de Vagrant sur la machine physique, ce qui nous a permis de créer un cluster local de VirtualBox VM utilisant l'image de base *dcos-vagrant-box*.

4.4.1 Installation DC/OS Vagrant

Après avoir cloner le repo Git officiel de DC/OS, nous avons lancé l'installation de laquelle a résulté la création de quatre machines virtuelles, deux agents (a1 et p1), une master (m1) et une boot. Contrairement à l'autre tentative, tout s'est déroulé correctement et nous avons pu accéder à l'interface GUI pour la configuration des services.

```
vagrant plugin install vagrant-hostmanager
git clone https://github.com/dcos/dcos-vagrant
cd dcos-vagrant
cp VagrantConfig-1m-1a-1p.yaml VagrantConfig.yaml
vagrant up
ci/dcos-install-cli.sh
```

Pour accéder en SSH à ces machines virtuelles, il suffit de lister les machines disponibles et de s'y connecter en utilisant Vagrant SSH.

```
vagrant status
vagrant ssh a1
```

4.4.2 Pods

Comme nous avons déjà une idée du fonctionnement de DC/OS, au vu de la première configuration, nous nous sommes tout de suite commencé par créer un pod. C'est-à-dire une *stack* elle-même composée de plusieurs containers. Dans ce pod, nous avons ajouté Resilio dans un premier container, puis Ngnix, PHP et Nextcloud dans un autre, comme représenté ci-dessous. Nous avons fait cela de cette manière pour qu'à chaque réplication du pod, les deux éléments puissent continuer à communiquer entre eux sans interférer avec les autres réplications, dans le cas où les containers auraient été répliqués séparément. Dans chacun des containers, nous avons créé un volume éphémère partagé afin de pouvoir partager les données de Resilio à Nextcloud. Les configurations de ces containers seront détaillées dans la suite de ce document.

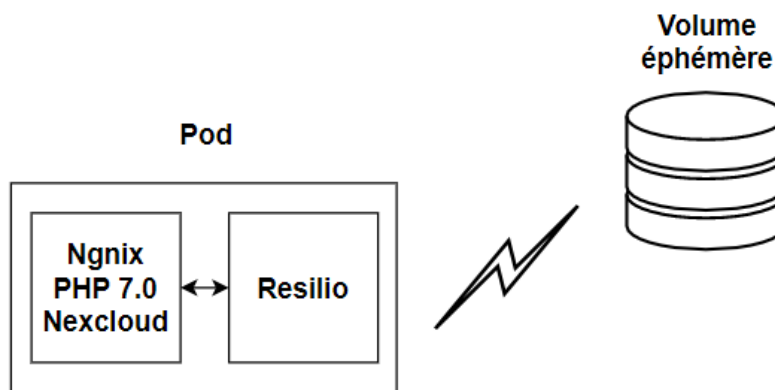


Figure 8: DC/OS pod

4.4.3 Resilio Sync

Suite à la première tentative de déploiement de Syncthing, nous avons réalisé la charge de travail pour faire fonctionner ce service de manière optimale. Il a donc été décidé de se tourner vers une autre application de partage de fichiers distribué. Nous avons connaissance d'un autre service nommé Resilio Sync mais, celui-ci n'étant pas open-source, nous avons préféré nous tourner en premier sur Syncthing.

Ce changement est justifié, car dans un des modes de fonctionnement de Resilio Sync, il est possible générer une clé unique pouvant servir d'identifiant unique à chacun de nos nœuds, là où Syncthing requiert un QR code pour chaque machine qui compte partager des données.

4.4.3.1 Création de l'image Docker

Nous avons été dans l'obligation de créer nous-même l'image Docker car celle proposée par Resilio/BitTorrent n'est pas compatible avec *Mesos engine*. Cette image ainsi que ses configurations peuvent être téléchargées sur notre repo⁶ Docker hub.

⁶ <https://hub.docker.com/r/qtask/sync-docker/>

Plusieurs images sont disponibles. Cependant l'image qui nous intéresse comporte le tag ":mesos". Ainsi, pour avoir l'image à disposition sur un ordinateur local, il faut simplement lancer la commande :

```
docker pull qtask/sync-docker:mesos
```

Les autres images ne sont que des étapes en vue de la construction finale de notre image. Ce sont majoritairement des images qui fonctionnent en docker engine mais pas forcément en *Mesos engine*.

Nous sommes finalement parvenus à un script de configuration⁷ très épuré. Partant de plus d'une vingtaine de lignes, nous sommes finalement arrivés à quelques lignes. Cela nous a permis de réduire considérablement l'empreinte de l'image qui n'est plus que de 66MB. Et tout ceci sans utiliser les « options » docker qui ne sont pas compatibles avec Mesos.

Cette configuration est assez explicite pour un utilisateur averti de linux, nous ne jugeons donc pas nécessaire de la commenter en détail.

Cette image ne comporte aucun "Entrypoint" qui, dans la terminologie des Dockerfile, indique un script à lancer automatiquement lors de la mise en service du container. Nous laissons la possibilité à l'utilisateur de lancer Resilio avec les commandes et arguments qu'il souhaite.

Cela nous permettra notamment de lancer le service *rs/sync* (Resilio) avec le PID user de *www-data* (33) se trouvant dans Ubuntu et plusieurs autres distributions l'utilisateur responsable des services web. De ce fait, cela assure la compatibilité avec notre autre service Nextcloud, notamment dans le partage des fichiers/volumes entre ces deux.

La commande typique pour lancer le container est :

```
docker run -rm -t someName -d -it qtask/sync-docker:mesos chown -R www-data:www-data /sync/ ; su -c \"rs/sync --nodaemon --config /sync/conf/sync.conf\" -s /bin/sh www-data
```

Le paramètre *--nodaemon* attribué à *./rs/sync* est nécessaire pour que l'interpréteur "tienne" sur le programme Resilio. En effet, un service docker est prévu pour quitter ou redémarrer quand un programme se termine. Si celui-ci est en mode démon, alors l'interpréteur peut difficilement savoir si le programme est toujours en exécution ou non, sans compter que le redémarrage ne doit pas être géré par l'hôte invité pour des raisons de gestion et d'audit entre autres. Comme un container est prévu pour un seul service dans 99% des cas, il est alors proscrit de lancer un service en tant que démon.

⁷ <https://github.com/qrzeller/sync-docker/blob/mesos/Dockerfile>

4.4.3.2 Implémentation du container dans DC/OS

Comme vous pouvez le voir sur la Figure 9, nous avons dû créer une configuration JSON spécialement conçue pour le pod Mesos. Nous avons eu du mal à trouver les clefs compatibles avec notre configuration de pod car celles-ci sont différentes de celles que l'on peut trouver avec les containers simples ainsi que les containers tournant sur le Docker Engine. Ici, nous trouvons uniquement la configuration du container. Les redirections de ports ainsi que le comportement général du pod seront décrites ailleurs. Ceci est donc une configuration partielle. Nous voyons cependant un certain nombre de choses qui sont le nom, utile aussi pour la résolution des noms DNS ainsi que les références inter-pod et l'allocation des ressources du container. Par contre, si le container n'arrive pas à s'exécuter du fait d'une allocation de ressources trop faible, celui-ci se relancera automatiquement avec plus de ressources. Ici nous avons pris de la marge. Nous voyons aussi que nous récupérons l'image de type Docker sur le docker-hub. Il est d'ailleurs également possible de créer des images de type RKT.

Il faut faire très attention de forcer le téléchargement des images. Avec Mesos, nous ne connaissons aucun moyen pour interagir physiquement / en ligne de commande avec les images Mesos. Nous comptons sur le fait que Mesos retélécharge toujours une nouvelle image avec l'attribut *forcepull* à *true*. Nous nous sommes aussi demandé ce qu'il advenait de tout l'historique des images, des logs et autres volumes orphelins. Il semblerait qu'un certain nombre de choses reste enregistré dans les méandres de Mesos. Un peu de recherche sur le web et les forums nous ont amené à la même conclusion. Il n'y a pas de moyen simple pour le non-averti de nettoyer le cache Mesos comme nous pouvons le faire sur Docker. C'est une question pour laquelle nous n'avons pas trouvé de réponses concrètes bien qu'il existe probablement une solution pour ceux qui utilisent ce système en production.

Nous montons un volume éphémère sur le chemin */sync/data/*. Ce type de volume s'apparente au container. Il est instancié et détruit en même temps que les containers. Sa construction physique est très semblable. Il existe plusieurs autres volumes. Les volumes hôtes (qui sont des mapping) les volumes *rexrey*⁸ spécialement créés pour les clusters et utiles pour la persistance des données créées par les containers.

⁸ <https://rexray.readthedocs.io/en/stable/>

```

1 {
2   "id": "/pds1",
3   "version": "2018-01-23T10:16:32.939Z",
4   "containers": [
5     {
6       "name": "resiliosync",
7       "resources": {
8         "cpus": 0.5,
9         "mem": 512,
10        "disk": 0
11      },
12      "image": {
13        "kind": "DOCKER",
14        "id": "qtask/sync-docker:mesos",
15        "forcePull": false
16      },
17      "exec": {
18        "command": {
19          "shell": "chown -R www-data:www-data /sync/\nsu -c \"rslsync \\\
20          \" --nodaemon --config /sync/conf/sync.conf\" -s /bin/sh www-data\"
21        }
22      },
23      "volumeMounts": [
24        {
25          "name": "test8",
26          "mountPath": "/sync/data/"
27        }
28      ]
29    }
30  ]
31 }

```

Figure 9 : script de lancement de resilio pour Mesos.

4.4.3.3 Problèmes rencontrés

Nous avons rencontré énormément de problèmes. Entre autres du fait qu'il n'y ait pas beaucoup de documentation sur internet. Il est quasi impossible de trouver des exemples fonctionnels de pod, hormis la documentation Mésosphère d'ailleurs tellement simplifiée qu'elle ne comporte pas d'image génératrice mais utilise l'image de base de l'hôte.

Nous avons donc dû faire nos propres expérimentations, premièrement, en passant par le Docker engine, pour enfin comprendre qu'il était encore impossible de créer des pod/stack en mode Docker engine. Nous sommes donc ensuite passés en mode Mesos engine, simple container, ce qui nous a permis de comprendre/découvrir les différentes fonctionnalités indisponibles. Encore une fois, cela nous a pris du temps car il n'y a pas d'interface de débbuging digne de ce nom pour cet engine. La moindre erreur ou beug remmène DC/OS à relancer indéfiniment le container sans indiquer quelques erreurs que ce soit. La seule mention d'indisponibilité de ses fonctions est non pas sur le site DC/OS, Marathon ou Mésosphère mais sur le site faisant l'inventaire des fonctionnalités de Mesos, au milieu de centaines de lignes de texte. Cela a donc été un travail très frustrant et laborieux. Nous savons donc maintenant que pour développer un projet sur une nouvelle plateforme dans un délais respectable il faut absolument regarder si l'outil est activement supporté par une communauté. Même si l'outil peut être très pratique, il est difficile de passer du temps à deviner la moindre étape ou à lire toute la documentation. Cette documentation n'est du reste pas très homogène entre les services mais aussi souvent disponible que dans le cache de google, et même de temps en temps contradictoire.

À noter que DC/OS possède une interface graphique pour la configuration des containers. Il est cependant peu recommandé de l'utiliser car celle-ci rajoute des bugs dans la configuration et rend inutilisables les containers (voir Figure 10). Sans comptes que certaines fonctionnalités ne sont pas prises en charge pour cette interface graphique.



Figure 10 : Bug ajouté par l'interface graphique.

4.4.4 Nextcloud

Pour ce service, nous avons pour objectif de mettre à disposition de l'utilisateur une interface web, pour qu'il puisse disposer de ces fichiers sans à avoir à passé par Resilio. Resilio dispose d'une clé unique et par conséquent nous ne voulons pas qu'elle soit partagée à l'utilisateur final.

4.4.4.1 Implémentation du container dans DC/OS

Nous avons également dû créer cette image de nous-même, la configuration du constructeur PHP étant incompatible avec le Mesos engine. Il a été plus difficile de créer cette image car elle dépend de beaucoup de scripts d'installation, de paramètres de sécurité etc... En explorant leurs scripts de construction d'image, nous nous sommes rendus compte tout de suite qu'il était impensable de partir sur ce chemin beaucoup trop laborieux et inefficace. Nous nous sommes ainsi rabattus sur le simple apt-get de Debian disposant d'une auto-configuration de beaucoup d'éléments. Mais ceci n'était pas gagné d'avance. Apt-get n'est pas réellement construit pour les scripts mais bien pour avoir une interaction utilisateur. Lors de l'installation PHP, il est demandé plusieurs paramètres de configuration de sécurité ainsi des configurations de modules. Nous avons donc au préalable dû mettre le *shell* en mode non interactif. Ceci étant encore insuffisant, nous avons changé quelques lignes de code du système. Cela était visiblement la solution la plus simple que nous ayons trouvée, la seule d'ailleurs sur certains forums. C'est le fichier *policy-rc.d*⁹ qui nous posait problème dans ce cas. En effet ce fichier décide ou non si le système peut lancer telle ou telle application en tant que services. Lors de l'installation de PHP, du fait que nous n'étions pas en mode interactif, le système/le script a décidé que celui-ci ne pouvait être lancé. Ainsi aucune configuration de PHP ne fut faite et le service était impossible à lancer. Une simple modification du code de sortie du script à 0 nous a permis de contourner facilement le problème, certes de manière peu élégante.

À cela, nous ajoutons une configuration type de Nginx communiquant avec le socket de PHP ainsi que l'installation, disponible sur cette page, de tous les modules nécessaires au bon fonctionnement de Nextcloud.¹⁰ Notamment la compatibilité avec MySQL, le processeur d'images, les modules de

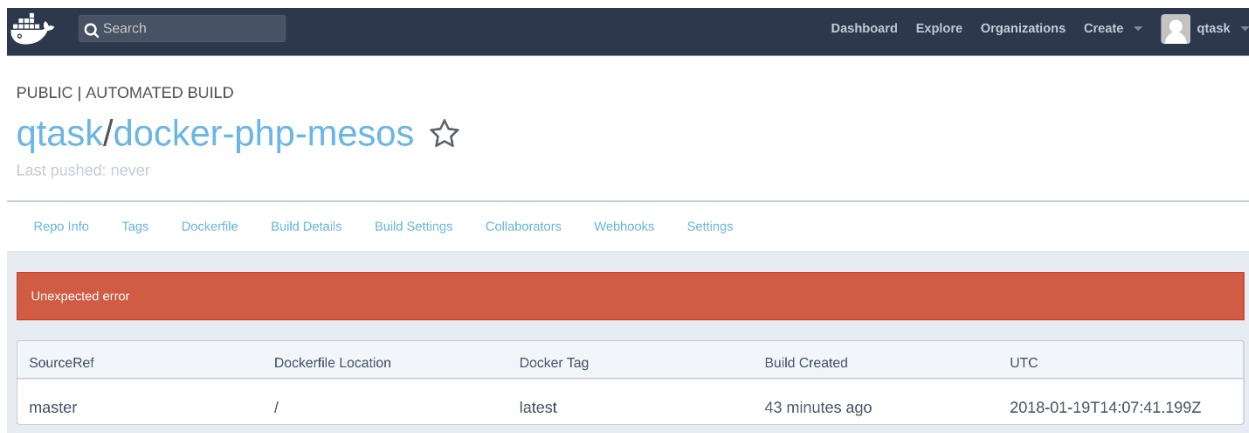
⁹ [http://www.chicoree.fr/w/Policy-rc.d_\(/usr/sbin/policy-rc.d\)](http://www.chicoree.fr/w/Policy-rc.d_(/usr/sbin/policy-rc.d))

¹⁰ https://docs.nextcloud.com/server/12/admin_manual/installation/source_installation.html

décompression, les modules de chiffrement, d'authentification, de cache, de génération de page et même de prévisualisation de vidéo.

4.4.4.2 Problèmes rencontrés

Nous voyons sur la Figure 11 que la construction de l'image sur docker hub a échoué sans raisons particulières, alors que celle-ci était parfaitement bien construite en local sur nos machines. Nous nous sommes rendus compte, après quelques investigations, que le problème venait de leur côté. En effet pour la construction des images docker, ils disposent d'une architecture virtualisée basée vraisemblablement sur un environnement similaire en partie à virtualbox. Et de ce fait, certaines actions pour la construction du docker ne sont pas supportées à l'exécution sur leurs machines. Ici, c'est simplement la commande mv qui, à l'interne, crée un *hardlink* du dossier que l'on veut mouvoir. Ce *hardlink* n'était pas géré au niveau matériel/virtuel sur leurs machines. Nous avons donc dû trouver une alternative.



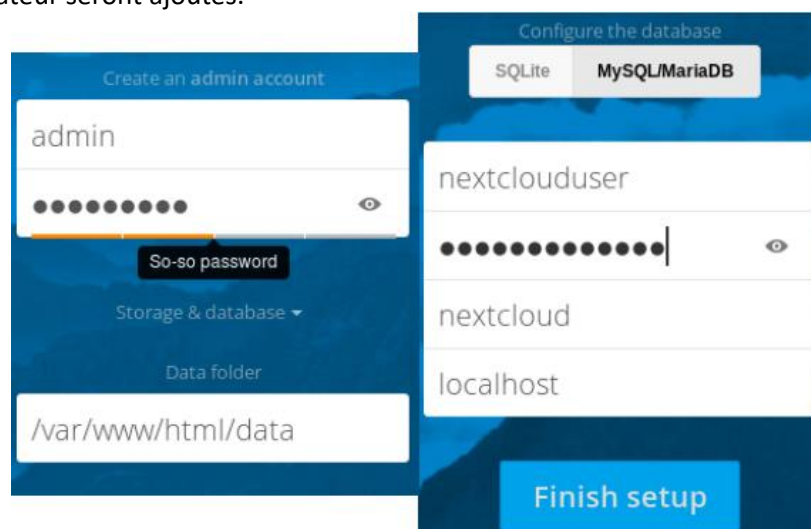
The screenshot shows the Docker Hub interface for the repository `qtask/docker-php-mesos`. The build status is "PUBLIC | AUTOMATED BUILD". The build details section shows an "Unexpected error" message. Below this, a table lists the build details:

SourceRef	Dockerfile Location	Docker Tag	Build Created	UTC
master	/	latest	43 minutes ago	2018-01-19T14:07:41.199Z

Figure 11 : Problème du build de l'image dans docker-hub

4.4.4.3 Interface web

Après avoir déployé l'image sur le cluster, lors de la première utilisation, il est nécessaire de créer un utilisateur, de spécifier le type de base de données, dans notre cas SQLite, et de définir le chemin où les fichiers de l'utilisateur seront ajoutés.



The screenshot shows the Nextcloud installation wizard. The first step is "Create an admin account" with fields for "admin" (username) and a password (masked with dots). Below the password field is a "So-so password" button. The second step is "Configure the database" with radio buttons for "SQLite" (selected) and "MySQL/MariaDB". Below this are fields for "nextclouduser" (username), a password (masked with dots), "nextcloud" (database name), and "localhost" (host). At the bottom, there is a "Finish setup" button.

Figure 12: Initialisation Nextcloud

Nous avons pour but de rendre cette opération transparente pour l'utilisateur afin qu'il ait juste à entrer ses identifiants pour accéder à ses fichiers. Nous avons généré une seule fois cette configuration et nous l'avons placée dans le dossier partagé avec Resilio. Ainsi, durant la réplication d'un pod, les fichiers seront ajoutés automatiquement. Nous avons réussi à effectuer cette opération. Cependant, lors de la génération initiale avec Docker en local, Nextcloud a rajouté cette IP pour que pas n'importe puisse accéder au serveur. Il était donc impossible d'accéder aux fichiers depuis le cluster.

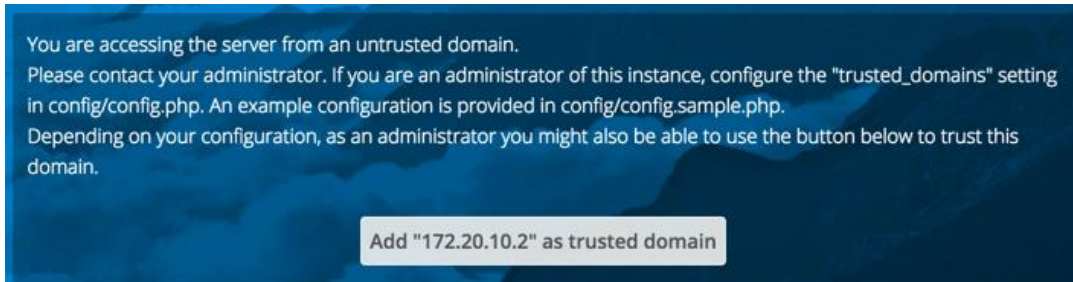


Figure 13: Erreur serveur IP Nextcloud

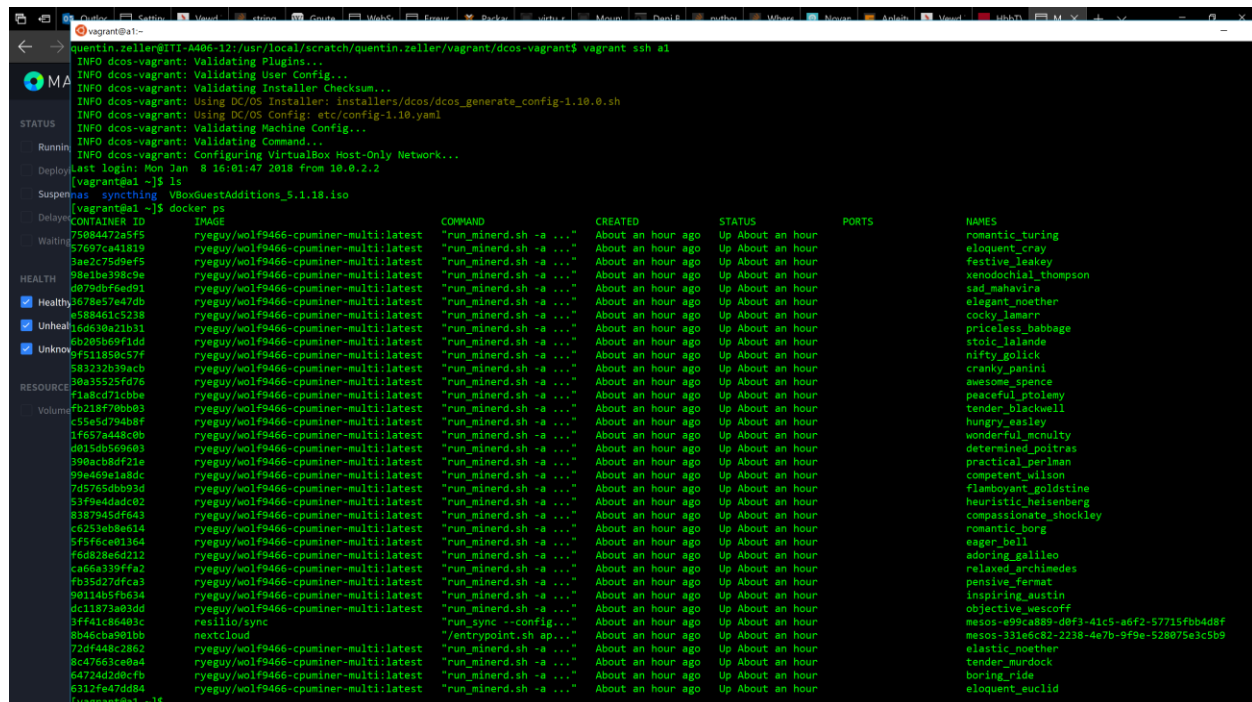
```
<?php
$CONFIG = array (
  'passwordsalt' => 'jyjvwLPyiemhsREtAVaylsgSyAr5DL',
  'secret' => 'inYKsHk9hZPQ1b/nXOEaopnmtIZX0CUY43FKrnnv8h8fEjCy',
  'trusted_domains' =>
  array (
    0 => '165.227.129.172',
  ),
  'datadirectory' => '/var/www/html/data',
  'overwrite.cli.url' => 'http://localhost',
  'dbtype' => 'sqlite3',
  'version' => '12.0.4.3',
  'installed' => true,
  'instanceid' => 'oc8hpn1xx7i5',
);
```

Figure 14: Résolution erreur IP Nextcloud

Nous avons aussi rencontré des problèmes avec les permissions, car Resilio ayant un PID différent, à la synchronisation aucun fichier était visible par Nextcloud.

4.5 Aparté

Comme vous pouvez le voir sur l'image dessous, quelqu'un s'est introduit dans notre système depuis notre interface web de gestion et a fait du *bitcoin/monero mining* sur nos machines. Il est probable que ce soit un étudiant. En effet, les ports utilisés par l'application sur nos machines ne sont pas les ports officiels de la distribution. Il est donc difficile pour un sniffer web de trouver ce genre de porte ouverte, bien que ce soit possible. Il s'agit de l'interface de gestion que nous avons ouvert pour le groupe de monitoring. Les personnes ayant installé ceci sur nos machines n'ont même pas dû faire quelques centimes. L'exécutable est imbriqué dans plusieurs couches de virtualisation en partant par VirtualBox. Même un smartphone serait plus performant pour ce genre d'opération hardware intensive nécessitant le cache des processeurs. Sans compter que ce sont des CPUs. Voir Figure 15.



CONTAINER ID	IMAGE	COMMAND	CREATED	STATUS	PORTS	NAMES
7584472e5f5	ryeguy/wolf9466-cpuminer-multi:latest	"run_minerd.sh -a ..."	About an hour ago	Up About an hour		romantic_turing
57697c04119	ryeguy/wolf9466-cpuminer-multi:latest	"run_minerd.sh -a ..."	About an hour ago	Up About an hour		eloquent_cray
3ae2c75d9ef5	ryeguy/wolf9466-cpuminer-multi:latest	"run_minerd.sh -a ..."	About an hour ago	Up About an hour		festive_leakey
98a1be398c9e	ryeguy/wolf9466-cpuminer-multi:latest	"run_minerd.sh -a ..."	About an hour ago	Up About an hour		xenodochial_thompson
d079dbfed91	ryeguy/wolf9466-cpuminer-multi:latest	"run_minerd.sh -a ..."	About an hour ago	Up About an hour		sad_mahavira
3678e57e47db	ryeguy/wolf9466-cpuminer-multi:latest	"run_minerd.sh -a ..."	About an hour ago	Up About an hour		elegant_noether
588461c5228	ryeguy/wolf9466-cpuminer-multi:latest	"run_minerd.sh -a ..."	About an hour ago	Up About an hour		coccy_lawarr
16d63ba21b1	ryeguy/wolf9466-cpuminer-multi:latest	"run_minerd.sh -a ..."	About an hour ago	Up About an hour		priceless_babbage
6b205b69f1dd	ryeguy/wolf9466-cpuminer-multi:latest	"run_minerd.sh -a ..."	About an hour ago	Up About an hour		stoic_lalande
9551185c57f	ryeguy/wolf9466-cpuminer-multi:latest	"run_minerd.sh -a ..."	About an hour ago	Up About an hour		nifty_golick
583232b39acb	ryeguy/wolf9466-cpuminer-multi:latest	"run_minerd.sh -a ..."	About an hour ago	Up About an hour		cranky_panini
30a35525fd76	ryeguy/wolf9466-cpuminer-multi:latest	"run_minerd.sh -a ..."	About an hour ago	Up About an hour		awesome_spence
71a8c071cbb6	ryeguy/wolf9466-cpuminer-multi:latest	"run_minerd.sh -a ..."	About an hour ago	Up About an hour		peaceful_ptolemy
f2b18f70b0d3	ryeguy/wolf9466-cpuminer-multi:latest	"run_minerd.sh -a ..."	About an hour ago	Up About an hour		tender_blackwell
155e5d794b8f	ryeguy/wolf9466-cpuminer-multi:latest	"run_minerd.sh -a ..."	About an hour ago	Up About an hour		hungry_easley
1f657a448c0b	ryeguy/wolf9466-cpuminer-multi:latest	"run_minerd.sh -a ..."	About an hour ago	Up About an hour		wonderful_mcnulty
d015db5696d3	ryeguy/wolf9466-cpuminer-multi:latest	"run_minerd.sh -a ..."	About an hour ago	Up About an hour		determined_poitras
398ac8d8f21e	ryeguy/wolf9466-cpuminer-multi:latest	"run_minerd.sh -a ..."	About an hour ago	Up About an hour		practical_perلمان
99e4091a8dc	ryeguy/wolf9466-cpuminer-multi:latest	"run_minerd.sh -a ..."	About an hour ago	Up About an hour		competent_wilson
7d5765db093d	ryeguy/wolf9466-cpuminer-multi:latest	"run_minerd.sh -a ..."	About an hour ago	Up About an hour		flamboyant_goldstine
53f9e4dad8c2	ryeguy/wolf9466-cpuminer-multi:latest	"run_minerd.sh -a ..."	About an hour ago	Up About an hour		heuristic_heisenberg
8387945df643	ryeguy/wolf9466-cpuminer-multi:latest	"run_minerd.sh -a ..."	About an hour ago	Up About an hour		compassionate_shockley
c6253e8e614	ryeguy/wolf9466-cpuminer-multi:latest	"run_minerd.sh -a ..."	About an hour ago	Up About an hour		romantic_borg
9f9f6ce01364	ryeguy/wolf9466-cpuminer-multi:latest	"run_minerd.sh -a ..."	About an hour ago	Up About an hour		eager_bell
76d828e0212	ryeguy/wolf9466-cpuminer-multi:latest	"run_minerd.sh -a ..."	About an hour ago	Up About an hour		adoring_gallio
c46a0339f2	ryeguy/wolf9466-cpuminer-multi:latest	"run_minerd.sh -a ..."	About an hour ago	Up About an hour		relaxed_archimedes
f35527d7fca3	ryeguy/wolf9466-cpuminer-multi:latest	"run_minerd.sh -a ..."	About an hour ago	Up About an hour		pensive_fernat
90114b5fb634	ryeguy/wolf9466-cpuminer-multi:latest	"run_minerd.sh -a ..."	About an hour ago	Up About an hour		inspiring_austin
dc11873a03dd	ryeguy/wolf9466-cpuminer-multi:latest	"run_minerd.sh -a ..."	About an hour ago	Up About an hour		objective_wescoff
3ff41c86403c	resilio/sync	"run_sync --config..."	About an hour ago	Up About an hour		mesos-e99ca889-d0f3-41c5-a6f2-57715fbb4d8f
804c0ba901b0	nextcloud	"/entrypoint.sh ap..."	About an hour ago	Up About an hour		mesos-331ec8c2-2238-4e7b-9f9e-528075e3c509
72d448c2862	ryeguy/wolf9466-cpuminer-multi:latest	"run_minerd.sh -a ..."	About an hour ago	Up About an hour		elastic_noether
8c47653ce0a4	ryeguy/wolf9466-cpuminer-multi:latest	"run_minerd.sh -a ..."	About an hour ago	Up About an hour		tender_murdoch
647242d0c1b	ryeguy/wolf9466-cpuminer-multi:latest	"run_minerd.sh -a ..."	About an hour ago	Up About an hour		boring_ride
6312fe47dd84	ryeguy/wolf9466-cpuminer-multi:latest	"run_minerd.sh -a ..."	About an hour ago	Up About an hour		eloquent_euclid

Figure 15 : Tentative de minage bitcoin/monero d'un intrus

5 Conclusion

Ce projet nous a permis de voir certains aspects de la virtualisation qui n'étaient pas connus ou pas totalement maîtrisés de tous les membre de ce projet. Nous pensons que ce projet était une bonne chose, car nous avons pu nous confronter à des problèmes bien réels pouvant survenir au sein d'une entreprise. Notre projet est fonctionnel. Cependant, il manque quelques fonctionnalités que nous aurions voulu mettre en place, telles qu'un DNS pour la communication inter-container/pod ou un serveur Nginx à répartition de charge (round robin) pour proposer à l'utilisateur le chemin le plus court pour l'accès à ce service.

6 Références

- <https://blog.docker.com/2015/11/scale-testing-docker-swarm-30000-containers/>
- <https://platform9.com/blog/kubernetes-docker-swarm-compared/>
- <https://www.upcloud.com/blog/docker-swarm-vs-kubernetes/>
- <https://docs.docker.com/engine/swarm/how-swarm-mode-works/services/#services-tasks-and-containers>
- <http://www.lemagit.fr/definition/Hyperviseur>
- https://members.loria.fr/lnussbaum/ptasrall2017/rf_dockerprod.pdf
- <http://www.lemagit.fr/conseil/Quelle-est-la-difference-entre-la-conteneurisation-et-la-virtualisation>
- <https://www.1and1.fr/digitalguide/serveur/know-how/conteneurs-informatiques-virtualisation-sans-emulation/>
- <http://www.lemondeinformatique.fr/actualites/lire-containers-ou-vm-comment-faire-le-bon-choix-64793.html>

7 Table des illustrations

Figure 1: Architecture Docker	6
Figure 2: Cluster Docker	7
Figure 3: Architecture de DC/OS.....	9
Figure 4: Espace DC/OS.....	10
Figure 5: Plan réseau partie 1	13
Figure 6: Web UI DC/OS.....	16
Figure 7: Exemple d'erreur sur DC/OS	17
Figure 8: DC/OS pod.....	19
Figure 9 : script de lancement de resilio pour Mesos.....	22
Figure 10 : Beug ajouté par l'interface graphique.	23
Figure 11 : Problème du build de l'image dans docker-hub	24
Figure 12: Initialisation Nextcloud	24
Figure 13: Erreur serveur IP Nextcloud.....	25
Figure 14: Résolution erreur IP Nextcloud.....	25
Figure 15 : Tentative de minage bitcoin/monéro d'un intrus.....	26