Table des matières

1 TABLE DES ILLUSTRATIONS

2 INTRODUCTION

2.1 CONTEXTE ET MOTIVATION

Les sciences de l'information apportent beaucoup au domaine médical. Elles permettent d'effectuer de nombreux calculs et algorithmes sur de grandes bases de données. Cela permet à de nombreux projets d'avoir facilement accès à l'information. Avec l'arrivée d'internet puis, aujourd'hui, du big data, de nouveaux enjeux sont apparus [1] [2] [3]. L'un d'entre eux est de rendre accessible ladite information. Un autre est de fournir un environnement de calcul puissant aux chercheurs dans un contexte protégé. Lesdites données sont souvent soumises aux closes de confidentialités médicales.

Une difficulté qui apparaitra souvent lorsque l'on cherche à fournir un tel environnement à des chercheurs est le contexte d'exécution. Aujourd'hui, il existe des dizaines de langages de programmation fonctionnant tous sur des compilateurs différents, et possédant tous des centaines de versions différentes. Nous rencontrons un réel problème pour déplacer et déployer facilement un écosystème dans les puissants centres de calculs.

Docker est une application permettant l'encapsulation d'un environnement à l'intérieur d'une boîte appelée conteneur. Celui-ci pourra ensuite être déplacé ou exécuté de manière simple, indépendamment du système hôte [3].

Cette solution se veut beaucoup plus légère aux machines virtuelles, déjà bien implantées sur le marché. Elle règle les problèmes liés au Cloud posés par les machines virtuelles. Ici, on ne déplace que l'environnement et uniquement lui. Là où une solution virtuelle prendra en compte un volume important de pilotes, de disques virtuels, ou encore de système d'exploitation [2].

En soi, nous prévoyons grâce à cette technologie de ne plus gérer des accès depuis l'extérieur. Maintenant, c'est le code qui doit se déplacer vers l'information, et plus l'inverse. Les bases de données contenant des informations confidentielles peuvent rester en sécurité derrière les pare-feux.

Le but de ce travail est d'explorer les solutions apportées par Docker dans le cadre d'une exécution à l'intérieur d'un centre sécurisé. Les images Docker sont créées par des chercheurs ou des médecins, et doivent être exécutées sur les serveurs.

Ce travail est réalisé dans le cadre d'un travail de Bachelor en Business Information System à la HES-SO Valais/Wallis.

2.2 OBJECTIFS

Le travail qui va suivre cherche à remplir un besoin : Fournir une plateforme permettant à un utilisateur extérieur d'exécuter son code avec son environnement sur les serveurs internes de l'institut de la HES-SO. Docker a été choisi pour permettre l'encapsulation dans un conteneur. Notre travail va être de chercher quelles sont les solutions qui permettraient de déplacer ces conteneurs, puis de les exécuter.

L'objectif principal sera donc de permettre à un utilisateur extérieur d'importer ses conteneurs dans le centre de donnée de l'institut. Il doit ensuite pouvoir être capable de déclencher l'exécution de son système, et en recevoir les résultats.

Un autre point important est de permettre aux conteneurs déplacés d'accéder à certaines ressources disponibles uniquement depuis l'intérieur. Dans ce cas précis, on fait surtout référence à des bases de données qui pourraient contenir des informations confidentielles. Seul le code se déplace. Les données ne doivent pas bouger.

La Figure 1 schématise ce que l'on cherche à atteindre. Les chercheurs envoient leurs conteneurs sur le cloud, représenté ici par un nuage. A l'intérieur, ils auront accès à toutes les données de tests. Leurs résultats leurs seront ensuite envoyés. Nous pouvons également voir dans ce modèle que les chercheurs disposent de données d'entrainement, pour tester leurs conteneurs avant de les envoyer.

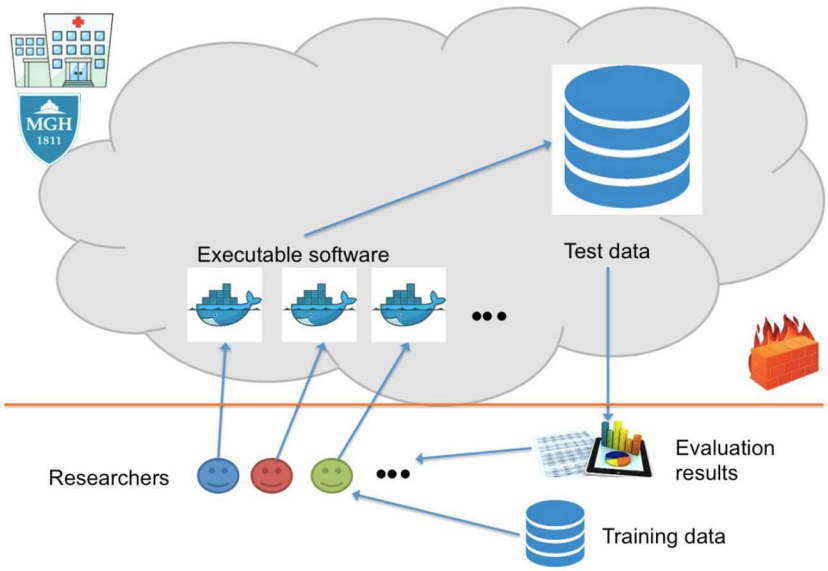


Figure 1 : Graphique objectif de la thèse

Un point qui pourrait être abordé dans le rapport serait la méthode qu'il faut utiliser pour recycler les conteneurs après leur exécution. Est-il possible d'intégrer un automatisme qui sera capable de supprimer les images après leur exécution ?

Un autre point qui pourrait être abordé est la sécurité non pas du transfert, mais de ce que le conteneur sera capable de faire une fois à l'intérieur. Existe-t-il une solution pour bloquer les appels étrangers qu'il pourrait émettre ?

Finalement, dans les points facultatifs, nous pourrons parler de la possibilité d'intégrer une interface utilisateur graphique au système et des possibilités de répartitions de charges sur Docker.

2.3 Résumé

Ce travail se focalise autour d'une technologie : Docker.

En préambule, nous ferons la liste de toutes les technologies qui seront utilisées plus tard. Ainsi, nous pourrons travailler en utilisant des outils connus.

La première partie consistera à la familiarisation avec Docker, sa psychologie, et ses modes de fonctionnement. Le but principal de cette étape sera surtout d'apprendre à encapsuler un environnement, puis d'exécuter un service à l'intérieur. Une bonne piste de départ serait de créer un service web entièrement portabilisé.

Ensuite, nous ferons également quelques essais avec l'API. Le but sera de voir comment elle fonctionne, et comment interagir avec.

L'étape suivante sera de rechercher comment déplacer les conteneurs à l'intérieur de l'infrastructure de manière sécurisée. On utilisera pour ce faire la puissance des technologies qui gravitent autour de Docker. Cette étape est la première concrète vers la solution que l'on recherche.

Après cela, nous rechercherons une méthode de contrôle à distance des images présentes en interne. Nous verrons qu'il n'existe pas de solution parfaite conçue à cet effet. Aussi, c'est pour cela que nous rédigerons une petite liste de possibilités et une comparaison entre elles.

Une fois nos recherches terminées, nous nous attarderons sur le déploiement des solutions trouvées sur l'infrastructure HES. Nous disposerons pour cela d'une machine virtuelle sur les serveurs, ainsi qu'un nom de domaine public.

Enfin, nous nous attarderons sur plusieurs points importants du travail qui n'ont pas pu être vus pendant la mise en place du projet. Nous étudierons les performances de Docker par rapport à une machine virtuelle et ferons le point sur les possibilités du GPU computing avec cet environnement. Nous verrons également un bref chapitre sur la sécurité.

Pour terminer, nous finirons sur une petite conclusion qui clora notre travail.

3 MÉTHODES

3.1 OUTILS DOCKER

3.1.1 Docker Engine

Docker est une plateforme open source permettant l'encapsulation de parties de logiciels dans un système indépendant du reste de l'ordinateur. Il sera donc possible d'y ajouter différents codes, librairies et logiciels sans pour autant modifier le système hôte. Cela fournit la garantie que l'exécution encapsulée dans Docker sera toujours la même, indépendamment de l'endroit où le conteneur sera exécuté [4].

Ce procédé est bien plus léger qu'une solution utilisant des machines virtuelles, puisqu'ici seuls les composants logiciels seront encapsulés. Un nouvel OS n'est pas nécessaire pour chaque application. Nous allons nous contenter d'encapsuler tous les composants nécessaires, et cela indépendamment les uns des autres.

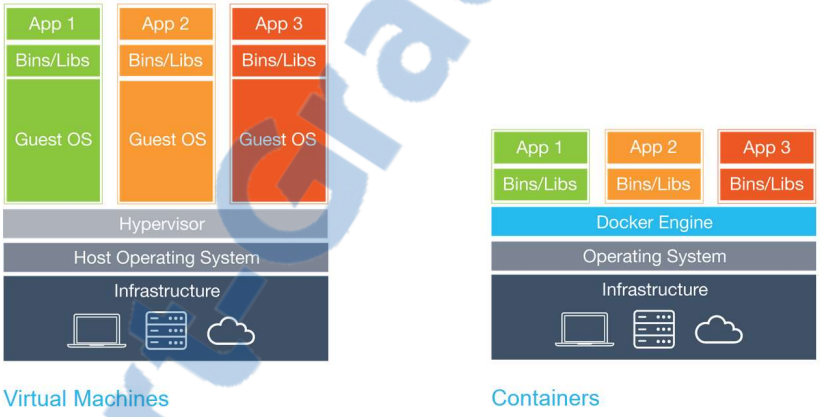


Figure 2: Comparaison VM vs Docker[1]

Il est à noter que Docker étant basé sur des technologies Linux, Il ne fonctionne que sur cet OS. Il est cependant possible de l'exécuter sur d'autres systèmes d'opérations grâce à un mélange entre machine virtuelle et Docker[2]. Au moment de la réalisation de ce travail, l'entreprise qui a créé Docker travaille activement sur une solution pour pouvoir l'utiliser sous Windows et OSX. Comme tout va très vite en ce moment dans ce domaine, et si le temps le permet, il sera peut-être possible d'essayer la beta du système sous Windows 10.

Le Docker Engine est le cœur de l'application. Il permet la création, l'exécution, et le lancement d'images Docker sur un système. Il s'agit donc de la pièce maîtresse du logiciel. Le contrôle de l'application se fait uniquement dans le terminal via des commandes personnalisées.

[1] https://www.docker.com/what-docker - Documentation officielle accédé le 13.06.16
[2] Voire chapitre « Docker Machine »

Avant de commencer à expliquer comment fonctionne Docker, il est important de comprendre la notion d'image et de conteneur. Une image est une « couche » contenant une configuration, ou un logiciel encapsulé. Les images sont organisées en pyramide, les unes sur les autres. Par exemple, si l'on veut créer une image d'un serveur Debian exécutant un serveur web, nous partirons de l'image Debian et y ajouterons une couche avec la version d'Apache désirée.

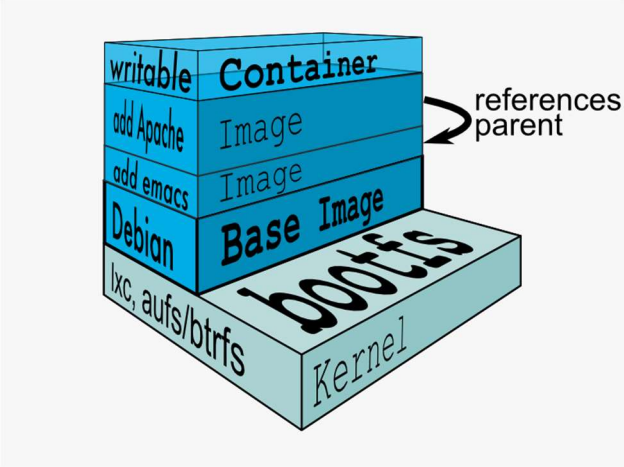


Figure 3 : Organisation en pyramide des images[3]

Pour définir une image, Docker se base sur un fichier de définition, appelé « Dockerfile ». Celui-ci doit contenir toutes les informations nécessaires pour la construction d'une nouvelle image. Lorsque nous construisons une image, nous effectuons un **Docker build**.

Notons également que Git, la solution de gestions de versions, fait partie intégrante de Docker Engine. Il est donc très facile de partager ses images via un hub centralisé[4].

Un conteneur se trouve au sommet de la pyramide des images. Il permet l'exécution de celles-ci. Là où une image n'est qu'une sorte de « modèle », le conteneur est un élément active. Il est d'ailleurs possible de lancer plusieurs conteneurs basés sur la même image. Quand on veut créer un conteneur, nous effectuons un **Docker run**.

Notez que si une image prend en compte toutes les modifications écrites à l'intérieur, ce n'est pas le cas des conteneurs. Au moment de l'arrêt, le conteneur est supprimé. Il faut donc être prudent lors de l'exécution de systèmes de données. Tout doit être enregistré en dehors du conteneur ! Il n'est pas difficile d'organiser cette liaison grâce aux volumes.

3.1.2 Les volumes

Pour pallier au problème de la persistance des données dans un conteneur, il existe une solution : Les volumes. Ils permettent de traverser l'environnement encapsulé et de faire un lien vers l'hôte. Cela veut dire que le conteneur partagera un de ses dossiers avec l'hôte. Le système est assez proche d'un partage de fichier standard.

Les volumes se déclarent lors d'un « Docker run » avec l'argument v. Il faudra ensuite simplement lier un dossier de l'hôte physique avec celui provenant du conteneur. Leurs contenus seront ensuite toujours identiques.

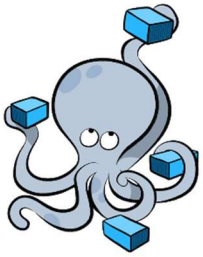
---

[3] https://www.docker.com/what-docker - Documentation officielle accédé le 13.06.16
[4] Voire le chapitre « Registry Server » qui s'étend plus sur ce point

Correctement utilisés, les volumes permettent de gérer la sauvegarde des données d'un conteneur et, dans notre cas, de lui fournir les informations nécessaires à ses calculs. Il est aussi possible de partager un même volume entre deux conteneurs pour leurs permettre de communiquer.

Pour ce travail, les volumes seront une fonctionnalité clé. Ils permettront de rendre les données des bases internes disponible dans l'environnement encapsulé.

3.1.3 Docker Compose



Docker Compose est un outil qui permet de définir et d'exécuter une application multi-conteneur[5]. Avec cette fonctionnalité, il sera possible de définir un fichier de composition qui permettra ensuite de lancer un environnement composé de plusieurs conteneurs.

Compose permet également la prise en charge de diverses options entrées normalement dans le terminal. Il permet donc également de simplifier le lancement grâce à ce plan d'exécution. C'est un outil très pratique pour définir la manière dont un environnement doit s'exécuter et les relations entre les conteneurs.

3.1.4 Docker Remote API

Docker Engine comprend une API (web services) proche du standard REST nommée Docker Remote API[6]. C'est elle qui permet la communication entre plusieurs applications exécutant Docker. C'est également elle qui va nous permettre de créer une interface de contrôle afin de maîtriser notre environnement encapsulé.

Dans ce travail, Remote API va nous permettre de gérer la communication entre nos éléments. Il s'agit d'un élément standardisé très ouvert qui va nous laisser beaucoup de possibilités.

3.1.5 Registry Server

Dans une utilisation classique de Docker, les images sont stockées directement sur l'hôte depuis lequel l'image a été construite. Cela ne permet pas d'assurer la pérennité de l'image.

Pour remédier à ce problème, la société Docker propose une intégration native à son Docker Hub. Il s'agit d'un serveur utilisant Git[7] permettant de sauvegarder ou de partager facilement des images sur le cloud. Malheureusement, ces images, sauvegardées sur le Hub, serons stockées sur les serveurs appartenant à Docker. Nous perdrons donc une partie du contrôle de ces images, puisque nous ne savons pas où elles sont stockées physiquement. Cela peut poser un problème du point de vue de la confidentialité.

---

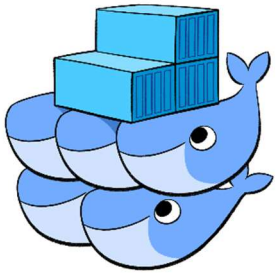
[5] https://docs.docker.com/compose/overview/ - Documentation officielle Docker Compose accédé le 13.06.16

[6] https://docs.docker.com/engine/reference/api/docker_remote_api/ - Documentation officielle Docker Remote API accédé le 13.06.16

[7] A la manière de GitHub.com ou BitBucket.org

Encore une fois, pour remédier à ce problème, nous avons la possibilité d'installer un serveur sur notre infrastructure. Il s'agit d'une fonction nativement disponible dans Docker depuis la version 1.6. Le Registry Server utilise la même configuration que Remote API et les fonctionnalités de Git pour déplacer des images.

3.1.6 Docker Swarm



Docker Swarm est une fonctionnalité native dans Docker permettant le clustering[8]. Il va donc permettre de réunir plusieurs hôtes Docker en un seul point d'exécution. Cette fonctionnalité va permettre de créer une infrastructure d'exécution unifiée. Elle est aussi très utile quand on a besoin de prendre en compte des éléments comme la montée en charge ou le temps de disponibilité.

Docker Swarm fonctionne grâce à l'API standard de Docker[9]. Cela le rend compatible avec tous les logiciels qui savent déjà communiquer avec le Docker Engine de manière transparente.

[8] https://docs.docker.com/swarm/overview/ - Documentation officielle Docker Swarm accédé le 13.06.16
[9] Voire chapitre « Docker Remote API »

3.1.7 Docker Machine

Pour la création et l'exécution de Docker sur Windows ou OSX, il existe un logiciel nommé « Docker Machine[10]». Celui-ci utilise les services Virtual Box[11] pour disposer des outils Linux nécessaires à Docker. Il s'agit avant tout de fournir une boîte à outils pour rendre Docker accessible depuis plusieurs plateformes.

L'installation par défaut comprend :

1. Docker Engine (donc Remote API ainsi que les fonctions de Registry)
2. Docker Compose
3. Docker Swarm
4. Certaines commandes docker-machine supplémentaire liées à l'hôte virtuel

Une fois que l'on utilise Docker Machine, il ne faudra pas perdre à l'esprit que tout nos conteneurs s'exécutent dans une machine virtuelle. Cela a comme conséquence majeure que certains pilotes de la machine physique seront inaccessibles et que l'interface réseau utilisée ne sera pas localhost mais celle de la vm. Les commandes docker-machine existent pour aider à gérer cette couche virtuelle supplémentaire.

Cependant, la machine virtuelle ne fournit que les fonctions d'exécution du kernel Linux. Le terminal Docker Machine donne l'illusion d'une exécution locale sur la machine physique. Ainsi, c'est le système de fichier de l'hôte qui sera utilisé. Une fois en ligne de commande, cela peut paraître un peu surprenant de se déplacer dans un système de fichier Windows à l'aide d'un terminal Linux.

[10] https://docs.docker.com/machine/overview/ - Documentation officielle Docker Machine accédé le 13.06.16

[11] L'utilisation d'autres logiciels de virtualisation avec Docker Engine, tels que VMware, est néanmoins possible

3.2 AUTRES OUTILS

3.2.1 Virtualisation VMware

Afin de disposer d'un environnement de développement local, nous allons utiliser la virtualisation à l'aide de VMware Workstation Player. Il s'agit d'un outil gratuit proposé par VMware, le leader mondial de la virtualisation.

Pour ce travail, nous aurons besoin de virtualiser un serveur pour effectuer nos tests. Cela nous permettra de simuler localement une infrastructure client-serveur, ce qui facilitera nos tests.

J'ai pris la décision d'utiliser les technologies VMware, car son concurrent, Virtual Box, est déjà utilisé par Docker Machine pour simuler les services Linux nécessaires. Nous aurons donc la preuve que notre système est indépendant. De plus, je possède davantage de connaissances dans cette technologie.

3.2.2 Jenkins avec plugin Docker

Jenkins[12] est une plateforme open source d'intégration continue. Fonctionnant sur des servlets Java, il intègre son propre serveur web et possède une interface graphique ergonomique. Jenkins est utilisé dans les entreprises voulant automatiser la compilation et le débogage durant le développement.

La grande force de ce projet open source est qu'il est modulable et possède de nombreux plugins. Nous pouvons trouver un plugin pour quasiment tous les besoins.

Pour notre projet, l'intégration continue n'est pas en soi un objectif que nous recherchons. Cependant, grâce au plugin Docker disponible sur Jenkins, il sera possible de profiter de la plateforme pour effectuer le pilotage de l'hôte.

En travaillant de concert avec la gestion des utilisateurs intégrée, Jenkins est une solution qui paraît toute indiquée.

3.2.3 Apache http Server

Apache est un serveur Web http open source[13] réputé pour être le plus populaire du World Wide Web[14]. Il permet l'hébergement de plusieurs services web sur un domaine.

Dans notre travail, Apache sera utilisé pour fournir des services à l'extérieur du réseau HES sans pour autant ouvrir toute l'API docker.

3.2.4 Web service PHP

PHP est un langage de programmation permettant principalement de créer des pages HTML. Il peut aussi être utilisé dans les programmes comme langage de scripts. Dans notre cas, il sera utilisé de concert avec Apache.

---

[12] https://jenkins.io/ - Site officiel de Jenkins accédé le 16.06.16
[13] http://httpd.apache.org/ - Site officiel du projet Apache http accédé le 13.06.16
[14] http://httpd.apache.org/ABOUT_APACHE.html - D'après son site officiel accédé le 13.06.16

Un web service est une méthode de communication utilisant le protocole http pour transférer des données entre deux applications. Ils sont de plus en plus utilisés pour gérer la communication entre deux applications. La forme d'un web service est décrite dans le modèle SOA.

Le but dans l'utilisation de PHP sera de créer un web service REST afin de piloter l'API Docker et de fournir certaines fonctions depuis l'extérieur. Nous allons développer une couche atteignable entre Docker et l'extérieur répondant à nos besoins spécifiques.

3.2.5 Nginx

Nginx est un serveur http concurrent d'Apache et un « reverse proxy ». C'est la seconde fonctionnalité qui va nous intéresser. Un proxy permet de filtrer le contenu web d'un utilisateur. Par analogie, un proxy inverse filtre ce qu'une personne située à l'extérieur peut faire à l'intérieur.

Nous allons donc utiliser Nginx pour filtrer les droits et autorisations des utilisateurs.

3.2.6 Serveur Linux

A la fin du projet, nous pourrons disposer d'un serveur dans l'infrastructure de l'institut afin de tester notre solution en conditions réelles. L'OS de ce serveur correspond en tout point à celui qui a été virtualisé. En outre, il sera possible d'y accéder via SSH via un outil tel que PuTTY[15].

Voici l'adresse de la machine dans le réseau : *vlhrbbigmed1.hevs.ch*

3.2.7 Certificat Let's Encrypt

Sécuriser les communications entre Docker et les clients est nécessaire. Pour ce faire, nous allons utiliser TLS. TLS est intégré dans Docker. Il a seulement besoin qu'on lui fournisse des certificats de sécurité. Une fois que cette sécurité a été activée, les commandes de l'API tronquent http par https.

La partie que nous allons sécuriser est la communication entre les clients et la registry. C'est dans cette communication que va se trouver le code qui pourrait être confidentiel. De plus, si nous souhaitons ensuite y ajouter une fonctionnalité de login, utiliser des communications sécurisée est un prérequis.

Afin d'obtenir des certificats, nous allons utiliser le service Let's Encrypt. Ce service permet de générer gratuitement des certificats. De plus, il permet l'entretient et la mise à jour des sécurités automatiquement, sans intervention humaine. Il arrive en effet fréquemment sur internet de rencontrer un site dont les certificats sont expirés, car ils doivent être renouvelé manuellement. Le projet Let's Encrypt permet de régler ce problème grâce à un système de challenge automatisé. L'intervention humaine n'est donc plus nécessaire. On peut noter qu'aujourd'hui, la plupart des navigateurs sont compatibles avec cette technologie.

Cette compatibilité touche notamment Docker. Nous allons donc en profiter pour générer nos certificats automatisés gratuitement.

Les seuls certificats qui seront généré seront destinés à la machine *vlhrbbigmed1.hevs.ch*

---

[15] PuTTY est un client SSH et Telnet populaire sur Windows. Il permet de se connecter facilement à une interface distante.

4 RÉSULTATS

4.1 PRISE EN MAIN DES OUTILS

4.1.1 Installation sous Windows 10

Docker pour Windows utilise une machine virtuelle Virtual Box sous Linux pour exécuter le cœur de ses images. Avant d'installer Docker, il est nécessaire de disposer d'une version récente d'Oracle Virtual Box. Virtual Box est disponible gratuitement depuis son site officiel[16].

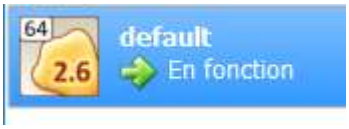
L'installation se fait ensuite depuis le site de Docker et est assez simple. Le site officiel fourni également un guide de l'utilisateur[17] pour démarrer rapidement, qui s'avère très pratique. Le composant qu'il nous faut installer se nomme Docker Machine. Celui-ci va utiliser Virtual Box, précédemment installé, pour nous fournir nos outils via un terminal.

La version actuelle de Docker pour Windows contient aussi Kitematic. Il s'agit d'une interface graphique permettant l'utilisation d'image Docker. Cette application est encore très limitée, car actuellement en alpha.

[16] https://www.virtualbox.org/ - site officiel de Virtual Box accédé le 13.06.16
[17] https://docs.docker.com/windows/ - Guide de démarrage rapide pour Windows accédé le 13.06.16

4.1.2 Docker Engine

Une fois l'installation de Docker Machine terminée, il est possible de lancer un terminal depuis l'application « Docker Quickstart Terminal ». Celui-ci utilise les ressources de la machine virtuelle appelée « default », crée par défaut lors de l'installation de Docker Machine. Si vous ouvrez Virtual Box, vous verrez que la machine est bien en fonction.



Un terminal s'ouvre ensuite. Il s'agit d'un terminal Linux réduit aux fonctions de bases Linux (tel que **ls** ou **cd**) et aux commandes Docker. Celui-ci est connecté à notre Windows. Il est donc possible de lister les dossiers de l'emplacement actuel (au démarrage le répertoire user connecté) ou de se déplacer dans l'arborescence.

Voici les versions de mes logiciels sous Windows dans la Figure 4 :

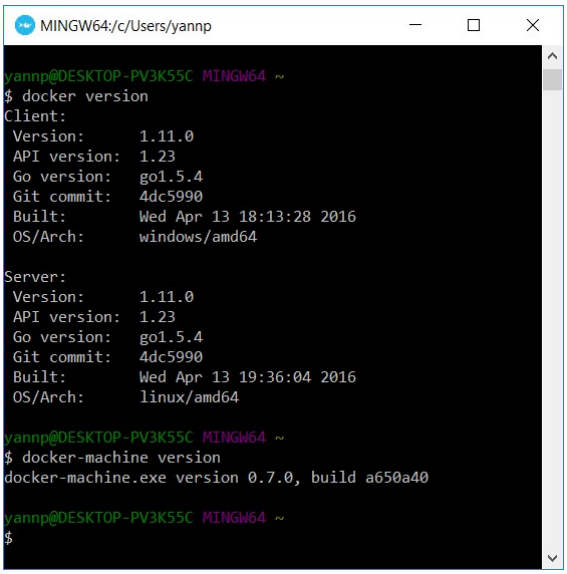


Figure 4 : Docker version sur la machine cliente

4.1.3 Provisionnement en image

Intéressons-nous maintenant aux images. Pour rappel, une image est une couche contenant, entre autres, des configurations et des librairies. C'est de ces images que l'on générera ensuite un conteneur [4] [5].

Pour commencer, Il va falloir commencer par créer un dossier quelque part dans l'arborescence puis y placer la commande du terminal. Il est possible de le faire directement depuis l'explorateur de fichier Windows ou depuis le terminal avec les commandes **mkdir** et **cd**.

Il faut ensuite écrire un fichier de définition de l'image nommé « Dockerfile ». Il doit s'appeler exactement comme cela, même la casse est importante ! Il ne doit pas non plus avoir une extension. Ce fichier contient la liste des commandes que Docker doit exécuter dans la nouvelle image.

Dans ce fichier, il faut commencer par dire de quelle image de base l'on part (directive **FROM**), puis on est libre d'ajouter ce que l'on veut. On s'intéressera surement à des commandes telles que **RUN** qui permettent de lancer une commande Linux dans le terminal de la nouvelle image. Par exemple, si nous souhaitons installer Nginx dans l'image, il faudra écrire :

```
RUN apt-get -y install nginx
```

Une fois ce fichier terminé, on peut créer l'image. C'est l'heure du Docker build !

```
docker build -t mon_image_1 .
```

L'argument t permet de spécifier un nom à l'image pour la retrouver plus facilement. Ensuite le point est important. Il spécifie où se trouve le Dockerfile à lire. Ici, nous nous sommes déjà déplacés dans le répertoire en question. Le point représente le répertoire courant.

Si nous créons des images inutiles par accident en environnement de test, voici une commande bien pratique qui va supprimer toutes les images sur l'hôte :

```
docker rmi $(docker images -q)
```

La commande rmi peut aussi être utilisée pour ne supprimer qu'une image. Parfois, il faudra également ajouter l'argument f pour forcer la suppression d'une image inutile.

Le chapitre suivant va étudier comment créer une image de serveur web et l'exécuter sous forme de conteneur.

4.1.4 Conteneur avec un serveur web

Pour cet exercice[18], voici le fichier que j'ai créé :

```
FROM ubuntu
# install updates if available and then nginx
RUN echo "deb http://archive.ubuntu.com/ubuntu precise main universe" > /etc/apt/sources.list
RUN apt-get update
RUN apt-get -y install nginx

#do some nginx configuration
RUN echo "daemon off;" >> /etc/nginx/nginx.conf
RUN mkdir /etc/nginx/ssl

#add the config file - the file default exist and is next to the Dockerfile
ADD default /etc/nginx/sites-available/default

# put a web page in the www folder - the file index.html exist and is next to the Dockerfile
RUN mkdir /var/www
ADD index.html /var/www/index.html

#Expose the port 80 to access the website
EXPOSE 80

#Run the websever
CMD ["nginx"]
```

Je suis parti de l'image Ubuntu. Il s'agit de l'image de base proposée par Ubuntu officiellement. Si elle n'existe pas en local, Docker va la télécharger automatiquement puisqu'elle est disponible sur Docker Hub et est entretenue par la communauté.

Voici un petit lexique rapide des commandes Docker utilisées dans ce fichier :

- FROM : Définit l'image de base depuis laquelle nous construisons la notre
- RUN : Lance une commande Linux durant le build
- ADD : Ajoute un fichier dans l'image depuis le système qui lance le Dockerfile
- EXPOSE : ouvre un port
- CMD : Définit l'action par défaut du futur conteneur. Seulement un disponible par image.

Docker entretient également un lexique des commandes pour Dockerfile sur son site officiel[19].

[18] Cet exercice est librement inspiré de https://serversforhackers.com/getting-started-with-docker accédé le 13.06.16

[19] https://docs.docker.com/engine/reference/builder/ - lexique officiel des commandes pour Dockerfile accédé le 13.06.16

Le build va ensuite aller installer les mises à jour, au cas où l'image Ubuntu présente localement ne serait pas à jour. Puis il passe à l'installation de Nginx. Nous constatons ici la puissance des commandes Linux utilisées avec Docker. Grâce à apt-get, il est possible d'installer n'importe quel logiciel. D'ailleurs, vu que nous avons accès au terminal de l'image, il est possible d'y mettre toutes les configurations que l'on souhaite.

Plus bas, grâce aux commandes ADD, on ajoute le fichier de configuration Nginx et le site web par défaut[20]. Puis, on informe Docker qu'il doit ouvrir le port 80 pour cette image lorsqu'un conteneur est lancé.

Fichier de configuration Nginx[21] (default):

```
server {
    root /var/www;
    index index.html index.htm;

    server_name localhost;

    location / {
        try_files $uri $uri/ /index.html;
    }
}
```

Finalement, et cela est l'un des éléments les plus importantes, il faut lui donner son « but ». Il s'agit de la commande CMD. Lorsque l'on voudra crée un conteneur à partir de cette image, Docker devra exécuter une commande dans le conteneur pour qu'il fasse ce pourquoi il a été créé. Ici, on lui dit de lancer la commande Nginx afin de démarrer les services web.

Nous pouvons maintenant déplacer le terminal avec cd dans le dossier et lancer le build de l'image. Docker va commencer par télécharger sur le Hub l'image Ubuntu de base, puis il va encore exécuter les commandes du Dockerfile.

Figure 5 : Téléchargement de l'image de base depuis le Hub

[20] Il s'agit d'un simple fichier HTML contenant le texte « This is Docker Nginx ! »
[21] Nginx n'étant pas un élément clé de ce travail, nous ne nous attarderons pas sur son fonctionnement

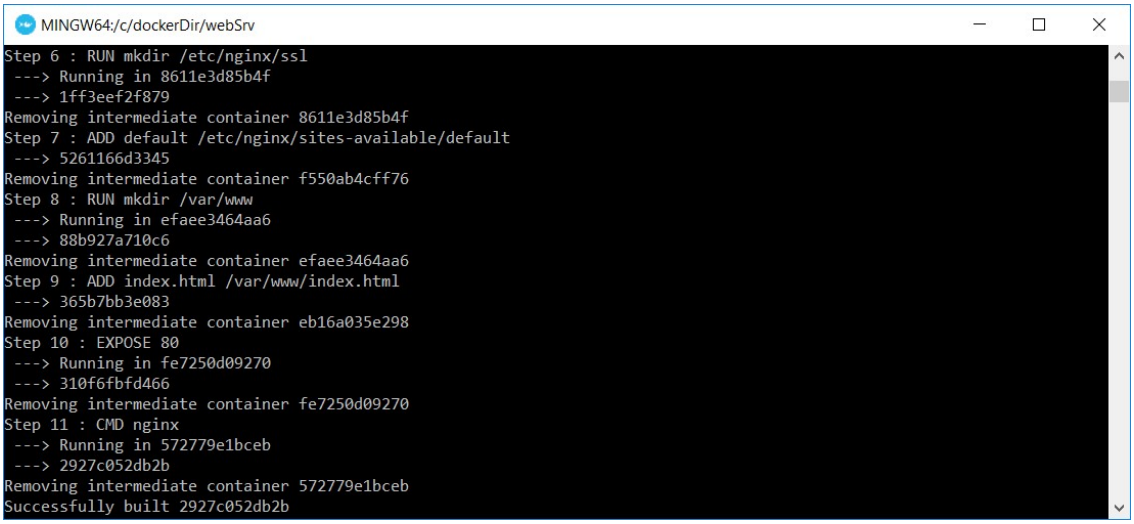


Figure 6 : Les étapes du Dockerfile exécutées indépendamment

Comme nous pouvons le voir sur la deuxième image, Docker crée des conteneurs pour lancer les commandes, puis les arrête immédiatement en ayant sauvegardé les changements dans des images intermédiaires. Il prend chaque étape (« step ») une par une !

Il est possible de visionner les images disponibles localement grâce à la commande docker images comme sur la Figure 7.

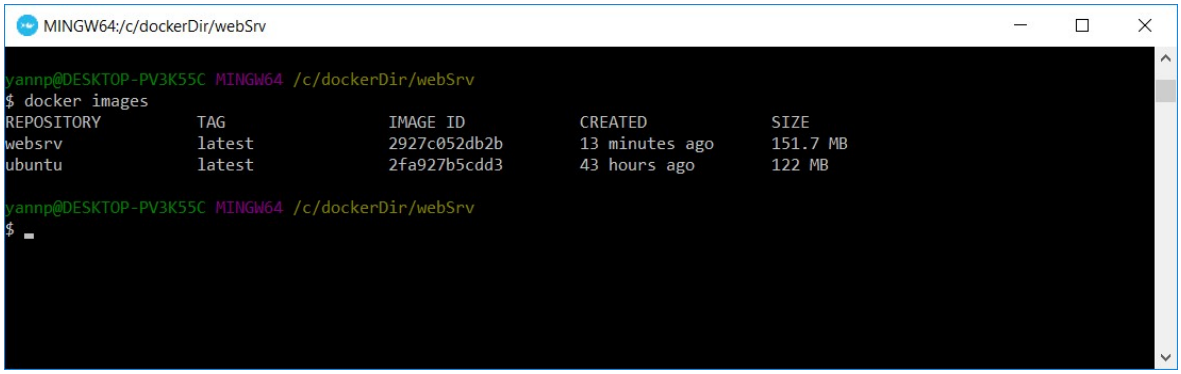


Figure 7 : Images disponibles après le build

On retrouve l'image Ubuntu téléchargée depuis le Hub et notre image nouvellement construite. L'image websrv ne fonctionnera plus si Ubuntu est effacé. On parle ici de dépendances entre les images. Il ne sera donc pas possible d'utiliser la commande rmi sur Ubuntu sans avoir effacé websrv au préalable[22].

Nous pouvons enfin exécuter notre conteneur !

```
docker run -p 80:80 -d test2
```

L'argument p est pour lier le port 80 de Docker à celui de la machine hôte (! Donc celui de la vm !). L'argument d est pour lancer l'exécution en mode détaché. C'est-à-dire que le conteneur tournera en arrière-plan et que l'on pourra toujours utiliser le terminal.

[22] Sauf si l'argument f (forces) est ajouté à rmi. Mais l'image websrv ne fonctionnera plus

La commande Docker ps affiche tous les conteneurs en exécution sur l'hôte comme on le voit sur la Figure 8.

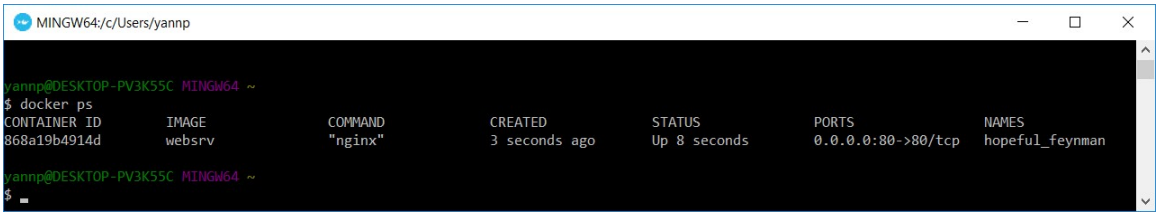


Figure 8 : Commande docker ps

Nous pouvons voir ici que le conteneur existe ! Chez moi, Docker l'a surnommé « hopeful_feyman » mais cela est aléatoire. Ce surnom peut être utile lorsqu'on lance plusieurs conteneurs à partir de la même image.

Nous pouvons maintenant accéder au site web, en tapant l'adresse IP de la machine virtuelle sous votre navigateur favori. Il est possible de trouver l'adresse IP facilement grâce à la commande docker-machin IP.

Figure 9 : Index du serveur web Nginx

4.2 DOCKER REMOTE API

4.2.1 Création d'un environnement de travail client - serveur

A l'heure où ce travail est créé, Docker n'est disponible que sur Linux (nativement) ou via Docker Machine (Virtual Box).

Afin de tester la communication entre les hôtes Docker, nous allons également créer une machine virtuelle qui va accueillir notre hôte « serveur ». C'est-à-dire qu'il va représenter l'hôte dans l'infrastructure avec lequel il va nous falloir interagir. Le Docker sur Windows, lui, représentera le client.

La machine virtuelle fonctionne sous VMware Workstation Player 12 par volonté de s'éloigner de l'infrastructure Virtual Box. Le Linux installé est Ubuntu. Le même OS sera probablement déployé plus tard sur le réseau HES-SO. Voici ses caractéristiques sur la Figure 10.

Figure 10 : Détails de l'environnement Serveur Linux

Sur ce serveur est installé Docker en version 1.11.1

```
root@ubuntu:/home/user# docker version
Client:
 Version:      1.11.1
 API version:  1.23
 Go version:   go1.5.4
 Git commit:   5604cbe
 Built:        Tue Apr 26 23:43:49 2016
 OS/Arch:      linux/amd64

Server:
 Version:      1.11.1
 API version:  1.23
 Go version:   go1.5.4
 Git commit:   5604cbe
 Built:        Tue Apr 26 23:43:49 2016
 OS/Arch:      linux/amd64
root@ubuntu:/home/user#
```

Figure 11 : Docker version sur le serveur

4.2.2 Initialisation de l'API

Docker Remote API permet d'accéder aux commandes de Docker depuis une machine distante. Il est nécessaire de l'activer.

En temps normal, pour rendre l'API accessible, il suffit de renseigner une variable d'environnement dans un fichier de configuration. Cela ne fonctionne malheureusement plus sur Ubuntu depuis la version 15.04[23]. Cela est dû à la migration récente de certaines distributions vers Systemd.

Une solution alternative est proposée, et c'est cette méthode qui fonctionne avec toutes les distributions et qui sera expliquée ici. Si elle ne vous convient pas, il est toujours possible de migrer votre système vers Upstart et d'utiliser les variables d'environnements Docker.

Il faut commencer par créer ce fichier :

```
/etc/systemd/system/docker-tcp.socket
```

Et le remplir avec ces informations :

```
[Unit]
Description=Docker Socket for the API
[Socket]
ListenStream=2375
BindIPv6Only=both
Service=docker.service
[Install]
WantedBy=sockets.target
```

Ainsi nous définissons notre propre socket qui pourra être utilisé par Docker. Notons bien que nous allons utiliser le port 2375 pour accéder à l'API.

Nous pouvons maintenant activer le socket grâce à ces commandes (à entrer dans le terminal) :

```
systemctl enable docker-tcp.socket
systemctl enable docker.socket
systemctl stop docker
systemctl start docker-tcp.socket
systemctl start docker
```

Voilà ! Nous pouvons maintenant accéder à l'API ! Nous pouvons déjà le faire en ligne de commande en local :

```
docker -H tcp://127.0.0.1:2375 images
```

L'argument H étant de donner une adresse IP distante pour l'exécution des commandes.

Essayons maintenant de le faire depuis l'hôte sur Windows !

---

[23] http://www.campalus.com/enable-remote-tcp-connections-to-docker-host-running-ubuntu-15-04/ accédé le 13.06.16

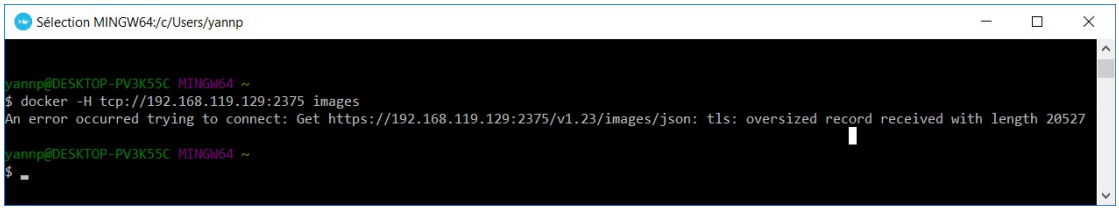


Figure 12 : Echec de l'appel de l'API depuis Docker Machine

Cela ne fonctionne pas … Cela est dû à une augmentation de sécurité intégrée dans Docker Machine : Il refuse de se connecter au web service sans https. Cela veut dire que si nous travaillions sous 2 machines Linux, il n'y aurait aucun problème.

Cependant, il est toujours possible d'appeler le web service de l'API avec une application capable de faire des appels REST. Ici, j'ai utilisé un plugin Firefox pour tester mon API. Nous pouvons le voire sur la Figure 13.

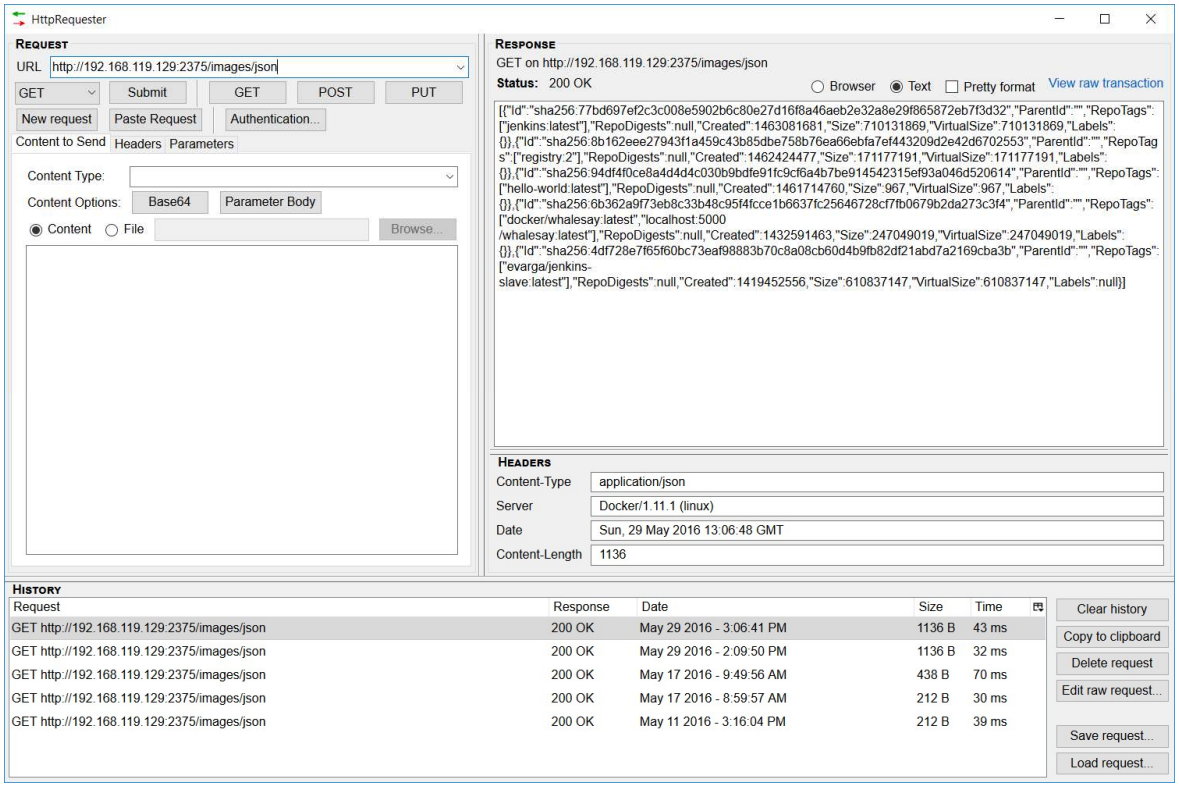


Figure 13 : Appel de l'API REST depuis un plugin Firefox

Nous appelons ici un fichier Json qui contient les mêmes infos que la commande Docker images.

```
http://192.168.119.129:2375/images/json
```

4.3 REGISTRY PRIVÉE

4.3.1 Création de la Registry

Plutôt que de créer une Registry depuis le début, nous allons utiliser l'image officielle[24] disponible sur Git. Cette image possède déjà toutes les fonctions voulues et persiste les images sur l'hôte grâce à un volume.

L'image est référencée dans l'index Docker, il est donc possible de lancer directement un Docker run et l'image sera automatiquement téléchargée depuis Github.

```
docker run -d -p 5000:5000 --restart=always --name registry   -v `pwd`/data:/var/lib/registry registry:2
```

La Registry fonctionne à présent et est disponible par le port 5000.

Observons un moment cette commande :

- L'argument d indique de lancer ce conteneur en arrière-plan. Ainsi on ne bloque pas notre terminal
- L'argument p attribue les ports entre le conteneur et l'hôte. Ici on lie les ports 5000 de l'un et de l'autre
- L'argument restart donne un comportement par défaut au conteneur au cas où le démon Docker s'arrête, comme lors d'un redémarrage par exemple. Ici on lui dit de toujours relancer ce conteneur.
- L'argument v est le plus intéressant. Il s'agit d'un volume. On lie le répertoire physique /home/data au répertoire encapsulé /var/lib/registry. Celui-ci contiendra les images qui vont être poussée dans notre conteneur. Si nous nous rendons sur Github et ouvrons le Dockerfile de l'image, nous verrons que le volume a bien été renseigné avant le build.
- Dernier point, nous nommons le conteneur « registry » avec la version 2. Nous utilisons bien la version 2 de la Registry docker.

4.3.2 Pousser une image

Nous n'avons pas encore d'image à pousser dans le Registry sur le serveur. Utilisons l'image d'exemple de Docker « Whalesay ». La commande suivante va télécharger l'image sur le Hub, va lancer un conteneur, le conteneur va écrire dans le terminal une image de baleine, puis le conteneur va s'arrêter. Ce qui compte réellement pour nous est de disposer d'une image à pousser dans la Registry.

```
docker run docker/whalesay cowsay boo
```

Nous allons maintenant tagger l'image Whalesay pour qu'elle puisse être envoyée dans notre Registry :

```
docker tag docker/whalesay localhost:5000/whalesay
```

[24] https://github.com/docker/distribution - image registry accédé le 13.06.16

La première partie du nom d'une image (avant le /) correspond en effet à l'emplacement de l'image. Maintenant que ce tag est mis, nous pouvons simplement effectuer un push dans la Registry.

```
docker push localhost:5000/whalesay
```

Figure 14 : Image poussée dans la Registry

Rendons-nous maintenant dans /home. Nous pouvons voire que le répertoire /home/data existe maintenant. Si nous explorons ce dossier, nous trouvons la Registry, puis à l'intérieur notre repository Whalesay !

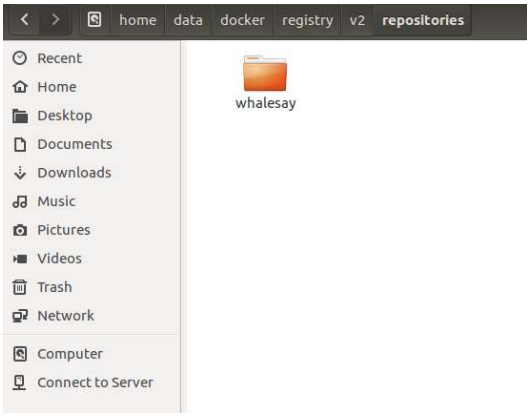


Figure 15 : Le repository contenant Whalesay

4.3.3 Utiliser les images stockées

Nous allons maintenant voire comment tirer une image du repository. C'est très simple. Il suffit de faire :

```
docker pull localhost:5000/whalesay
```

Et l'image stockée sur le repository sera copiée en local.

Mais cela n'est pas le plus intéressant : La commande Docker run est préconfigurée pour lancer un pull des images qu'elle ne trouve pas. Ainsi, si on lance un run d'une image qui n'est pas présente en local, docker ira la chercher dans le repository !

Pour en être certain, j'ai coupé le réseau lors de ce test.

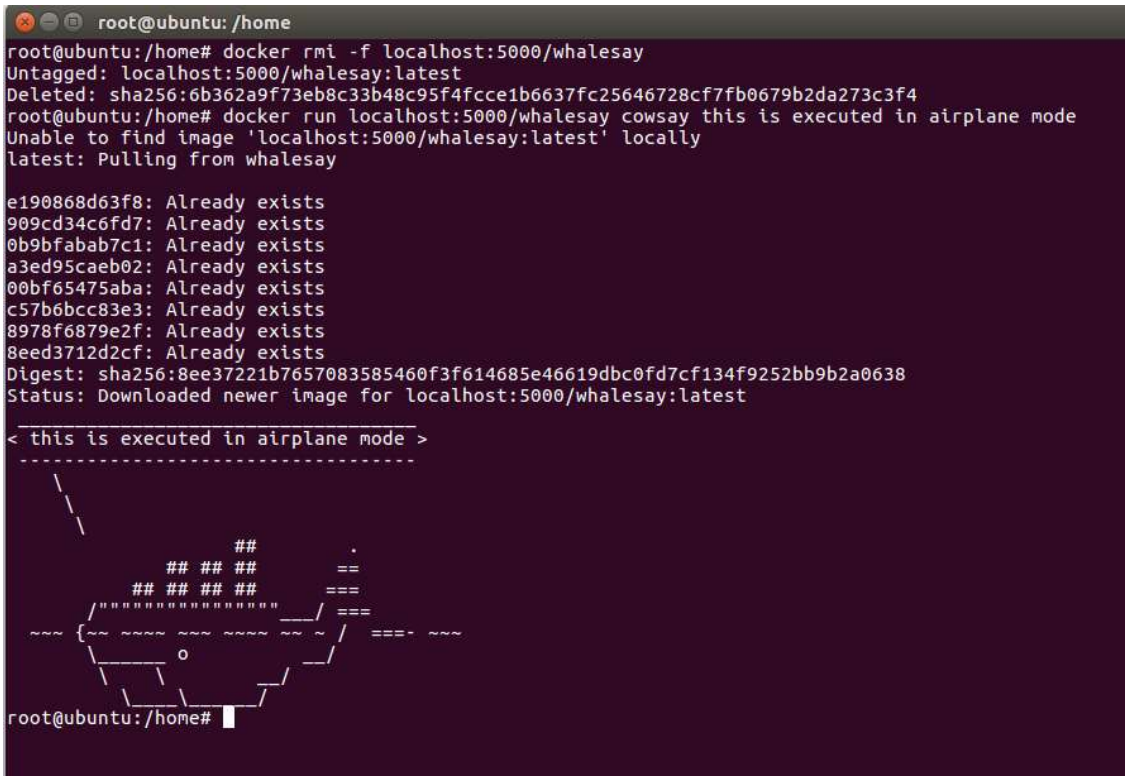


Figure 16 : Image lancée sans connexion réseau

4.4 RECHERCHE D'UNE MÉTHODE DE CONTRÔLE

Dans ce chapitre, nous aborderons les différents moyens qui existent pour lancer un conteneur sur un hôte distant. En combinant les forces de la registry (enregistrement des images dans l'infrastructure) et de l'API, nous pouvons envisager plusieurs solutions pour résoudre notre besoin.

4.4.1 Plateforme de CI avec Jenkins : installation

Il est possible avec Docker de créer une plateforme d'intégration continue en utilisant Jenkins. Celui-ci possède un module développé spécifiquement pour interagir avec l'API de Docker (Remote API).

Dans cette partie du travail, nous allons déployer une plateforme Jenkins sur notre serveur Docker afin de voir s'il comprend les fonctionnalités voulues pour notre projet.

Une image Jenkins officielle existe sur Docker hub. Nous pouvons donc effectuer directement un run.

```
docker run -d -p 8080:8080 jenkins
```

- Le port 8080 correspond à l'interface web

Jenkins est maintenant accessible.

Figure 17 : Interface web Jenkins

Il faudra ensuite se rendre dans l'outil de gestion des plugins et y ajouter le plugin Docker

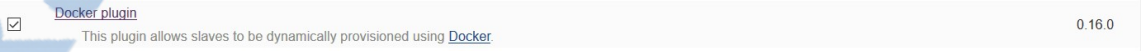


Figure 18 : Plugin Docker Jenkins - Download

Malgré que cela ne soit pas précisé dans la documentation du plugin, les fonctions de lancement de conteneur nécessitent également « Token Macro Plugin ».

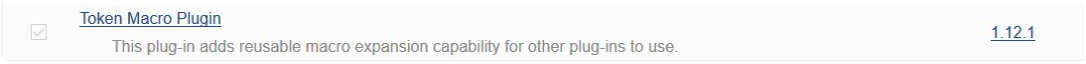


Figure 19 : Token Macro Plugin

Une fois ces deux plugins installés, il faut ajouter l'hôte Docker comme référant Cloud dans la configuration de Jenkins. L'option cloud se trouve tout au fond de la configuration du système.

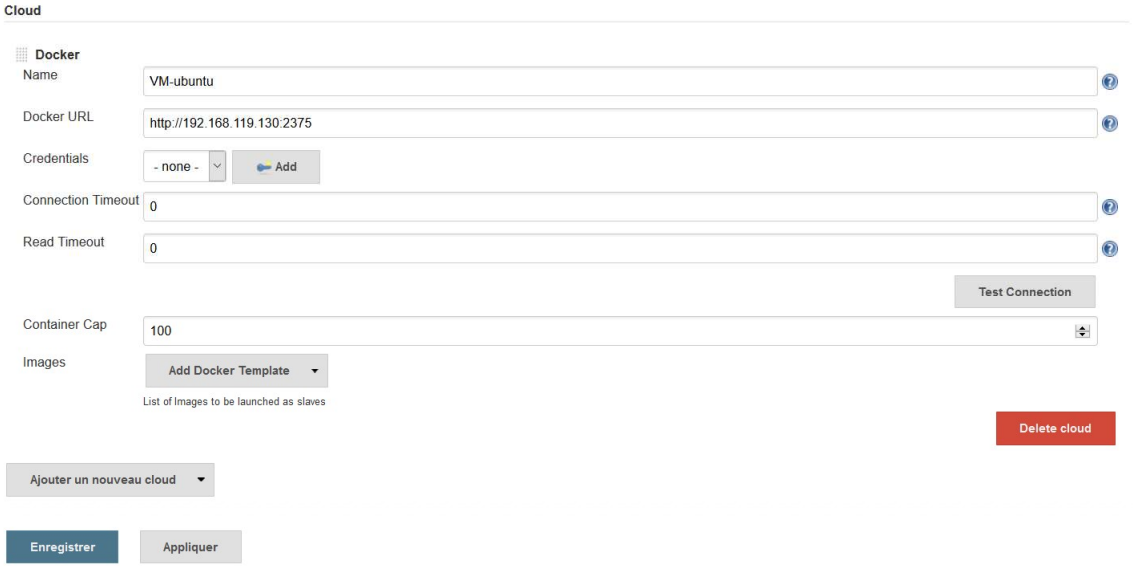


Figure 20 : Configuration d'un fournisseur de Cloud sur Jenkins

Il y serait même possible d'y ajouter plusieurs hôtes.

Une petite interface de pilotage est maintenant disponible dans les paramètre du système, menu Docker.

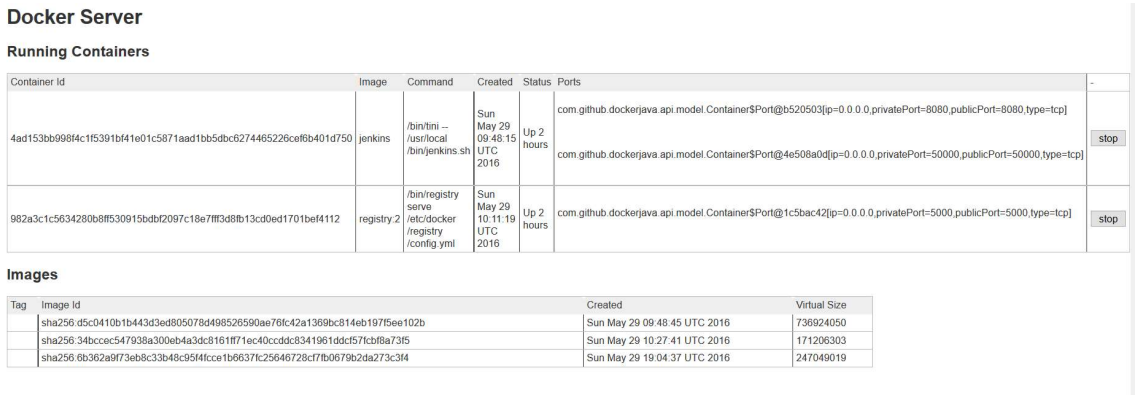


Figure 21 : Interface de pilotage de l'hôte sur Jenkins

Elle reste très limitée, mais est toujours pratique.

N'oublions pas que toutes les configurations faites sur Jenkins ne sont présente que dans le conteneur. Si on l'arrête par erreur ou suite à un arrêt de l'hôte, il faudra retrouver le conteneur exacte grâce à la commande :

```
Docker ps -a
```

Qui affiche tous les conteneurs même ceux arrêté. Puis effectuer un Docker start sur le bon. Attention, un run créera un nouveau conteneur et nous ne récupèrerions pas nos configurations !

Il est recommandé d'enregistrer les modifications dans une nouvelle image. Pour ce faire, on utilise la commande :

```
Docker commit <container ID> jenkins-yann
```

Cela créera une nouvelle image localement disposant des paramétrages.

4.4.2 Plateforme de CI avec Jenkins : utilisation

Pour lancer un conteneur, il faudra ensuite crée un nouvel item. Une item correspond à un projet dans le langage Jenkins. Il faudra ensuite crée un projet de type freestyle. Ce type nous permets de choisir librement ce que l'on souhaite faire. Comme nous voulons simplement lancer un conteneur, c'est le meilleur type de projet. Un projet classique Jenkins verrait plutôt la prise en charge d'un outil de build tel qu'Apache Ant. Nous n'allons pas nous en servir.

Dans la configuration du projet, nous n'allons rien modifier. Seulement ajouter comme étape de build le run d'un conteneur.

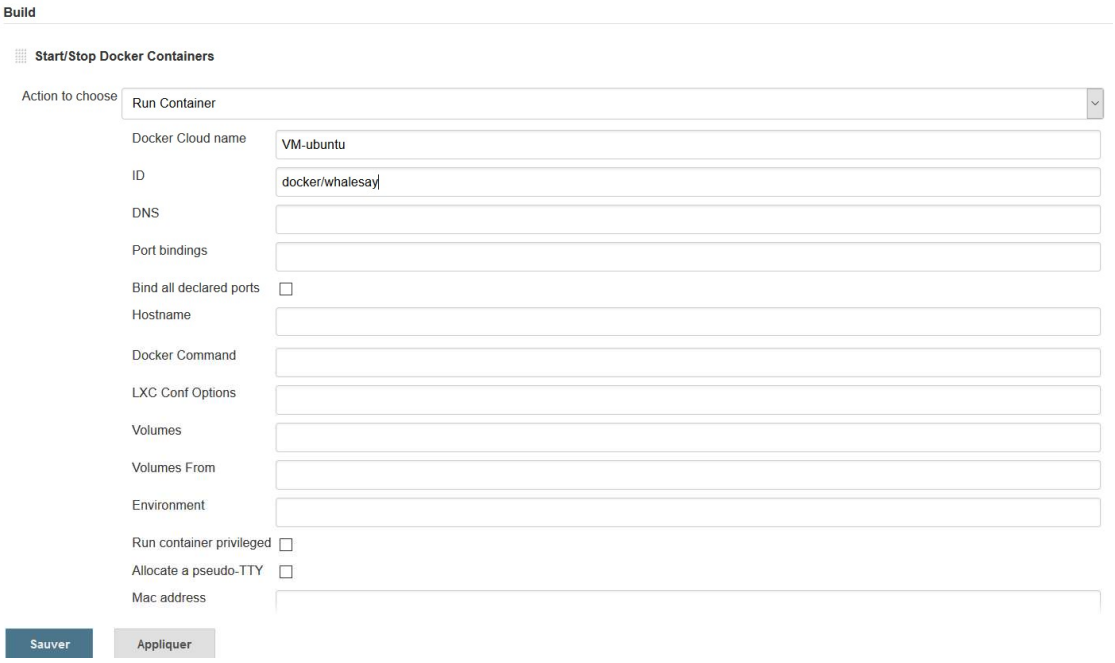


Figure 22 : Configuration d'un Docker Run sur Jenkins

Comme ce type de projet correspond à la commande *Docker run* le système va s'occuper de tout le nécessaire. Pour rappel, un run sur Docker correspond à toute une série de commande allant du pull jusqu'au retour des logs. Inutile donc de configurer plus de chose.

Nous constatons également que ce run est paramétrable. Pour notre projet, c'est la ligne « volumes » et la ligne « Ports bindings » qui nous intéressent spécialement.

Pour lancer le projet, il suffit ensuite de lancer le build. L'historique des builds apparait ensuite sous le menu principal, dans les paramètres du projet. A l'intérieur, on trouve le menu « Console Output » qui montre ce que la console Docker a affiché pendant le run.

Figure 23 : Output de la console Docker dans Jenkins

Cet historique gère aussi tous les précédents lancements de conteneurs. Il est ainsi possible de consulter l'historique de ce qui a été fait sur le serveur.

Ce système comprend un défaut majeur : Il ne fournit pas les logs émis par le conteneur pendant son exécution. Une image tel que Whalesay, par exemple, qui écrit son résultat dans le terminal, ne pourra fournir son retour.

4.4.3 Plateforme de CI avec Jenkins : Gestion des utilisateurs

Notre plateforme pour piloter notre hôte est déjà en ligne ! Elle souffre cependant d'un gros problème : Tout le monde peut se connecter sur le site et faire ce qu'il désire. Nous allons donc travailler sur un moyen pour protéger Jenkins. Nous allons également en profiter pour créer un système d'utilisateur poussé.

Nous allons utiliser le plugin « Role-based Authorization Strategy »

Figure 24 : Plugin Jenkins pour la sécurité sur les rôles

Après avoir installé le plugin, il faut se rendre dans le menu de la sécurité globale de Jenkins et l'activer. Le plugin est disponible dans les choix de méthode d'autorisation.

Nous pouvons noter qu'hormis les utilisateurs standard sur Jenkins, il est possible d'intégrer certaines bases utilisateur d'autre système. On notera particulièrement les bases d'utilisateur UNIX, c'est-à-dire les comptes des systèmes Linux, et le LDAP, qui correspond aux bases d'utilisateurs Windows. L'option « Déléguer aux servlets » permet à l'utilisateur avancé de déléguer la gestion des utilisateurs au moteur Java qui exécute Jenkins.

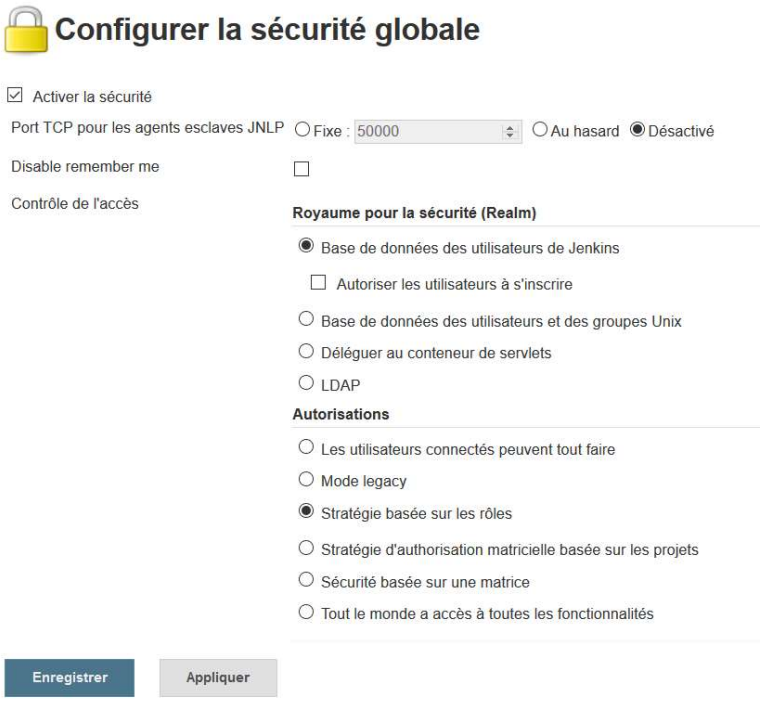


Figure 25 : Configuration de la sécurité globale sur Jenkins

Par défaut, l'utilisateur anonyme est dans le groupe « admin » et possède tous les droits. Toute personne qui accède au site sans se loguer est connecté en tant qu'Anonyme. Il est donc important de créer un nouvel utilisateur admin et de lui donner les droits administrateurs. N'oubliez pas ensuite de supprimer les droits sur Anonyme. Maintenant, on ne peut plus accéder à la plateforme sans se connecter.

Attention cependant, si vous coupez les droits de l'utilisateur Anonyme avant d'avoir créé votre administrateur, il ne sera plus possible d'accéder à Jenkins et vous pourrez recommencer depuis votre dernière image.

Le plugin de l'authentification basée sur les rôles nous est très utile car il permet de :

- Définir des droits sur un projet : Grâce à cette fonctionnalité, il est possible de donner des autorisations directement sur un projet. Très utile si l'on souhaite que certains utilisateurs ne puissent voir que les projets qu'ils managent.
- Définir des groupes : Regrouper plusieurs utilisateurs sous un même groupe. Plusieurs utilisateurs pourront donc travailler sur le même projet.
- Définir des rôles : Indépendamment des groupes, il sera possible de définir quels sont les casquettes de chaque utilisateur.

4.4.4 Plateforme de CI avec Jenkins : Synthèse

Nous avons vu au fil de cette installation que Jenkins offrait une plateforme évoluée dans l'intégration continue. Son management poussé des utilisateurs et des projets en fait une zone d'administration de projet de premier choix. De nombreux plugins complètent les fonctionnalités présentes nativement. C'est en cela que nous parvenons à atteindre notre hôte Docker.

Malheureusement, la solution Jenkins, même si elle reste assez pratique, comprend plusieurs défauts que l'on ne peut ignorer si l'on souhaite ouvrir son accès à l'extérieure :

- De nombreuses fonctions de configuration de projet peuvent être configurée par le client et on ne peut l'interdire de le faire. C'est toujours compliqué de devoir dire à l'utilisateur d'ignorer une grande partie des champs qu'il pourrait vouloir remplir dans la configuration d'un projet
- Aucune gestion des retours des logs de l'exécution. Il faudrait compléter avec autre chose.

Malgré ces défauts, la solution Docker sur Jenkins a le mérite d'offrir une bonne plateforme de CI interne. Son utilisation serait très adaptée si elle était utilisée en interne avec des gens qui savent ce qu'ils font. Pour l'extérieure, en revanche, il faudra se pencher sur une autre solution.

4.4.5 Shipyard

Un projet Open Source pour faciliter le contrôle d'un hôte Docker existe : Shipyard[25]. Reposant sur une plateforme web, Shipyard permet de voir très facilement ce qui se passe sur l'hôte grâce à une interface graphique dédiée.

Pour voire les possibilités de ce programme, nous avons copié la machine virtuelle Ubuntu, afin de ne pas devoir démonter tout ce qui a déjà été entrepris. Ensuite, il nous faudra arrêter le socket crée lors de l'ouverture de l'API (docker.socket). L'installeur de Shipyard va en recrée un dédié qui utilisera le même port.

Pour lancer l'installation :

```
curl -sSL https://shipyard-project.com/deploy | bash -s
```

Shipyard s'installe ensuite automatiquement. Le logiciel et ses dépendances sont présents dans des conteneurs que l'on voit sur la Figure 26.

```
root@ubuntu:/home/user# docker ps
CONTAINER ID        IMAGE                              COMMAND                  CREATED             STATUS              PORTS                                                NAMES
fd7805a482bf        registry:2                         "/bin/registry serve "   2 weeks ago         Up 2 weeks          0.0.0.0:5000->5000/tcp                               registry
e14117edc614        shipyard/shipyard:latest           "/bin/controller --de"   2 weeks ago         Up 2 weeks          0.0.0.0:8080->8080/tcp                               shipyard-controller
9569ef6fbd1e        swarm:latest                       "/swarm j --addr 192."   2 weeks ago         Up 2 weeks          2375/tcp                                             shipyard-swarm-agent
c79ee1ba7bc3        swarm:latest                       "/swarm m --replicati"   2 weeks ago         Up 2 weeks          2375/tcp                                             shipyard-swarm-manager
5ec9eecaad79        shipyard/docker-proxy:latest       "/usr/local/bin/run"     2 weeks ago         Up 2 weeks          0.0.0.0:2375->2375/tcp                               shipyard-proxy
62415f01845f        alpine                             "sh"                     2 weeks ago         Up 2 weeks                                                               shipyard-certs
a1c72957d5c2        microbox/etcd:latest               "/bin/etcd -addr 192."   2 weeks ago         Up 2 weeks          0.0.0.0:4001->4001/tcp, 0.0.0.0:7001->7001/tcp       shipyard-discovery
f1b8c84d227b        rethinkdb                          "rethinkdb --bind all"   2 weeks ago         Up 2 weeks          8080/tcp, 28015/tcp, 29015/tcp                       shipyard-rethinkdb
root@ubuntu:/home/user#
```

Figure 26 : Conteneurs Shipyard

L'interface web est accessible par défaut sur le port 80 de l'hôte.

[25] https://shipyard-project.com/ - site officiel de Shipyard – accédé le 13.06.16

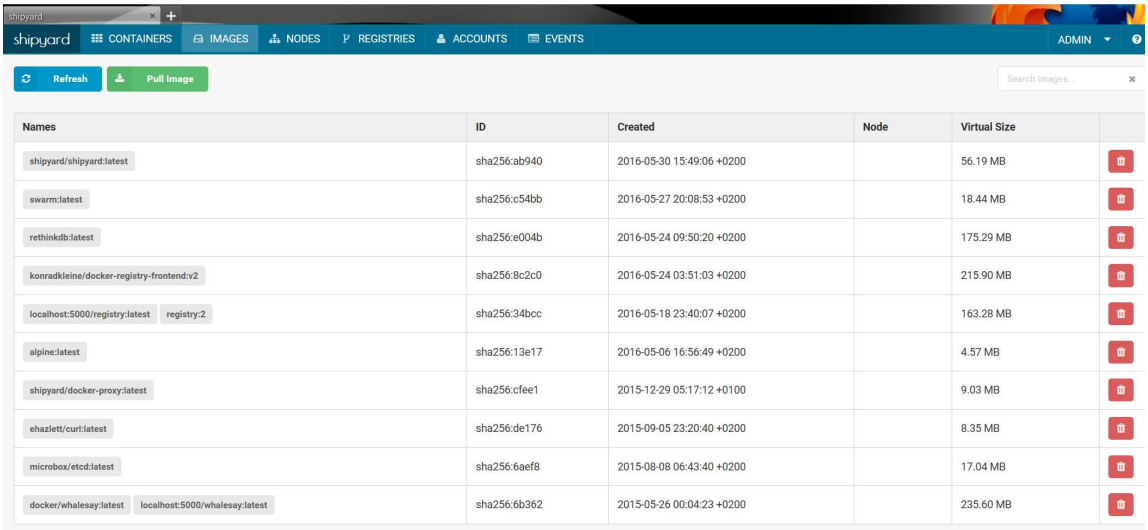


Figure 27 : Interface Shipyard

Chaque menu permet d'accéder à une fonction différente et de piloter l'hôte de manière précise. Nous pouvons consulter la liste des images, des conteneurs, etc. Nous pouvons aussi les démarrer, les supprimer ou effectuer toutes autres fonctions. Shipyard est très pratique pour ce qui est du pilotage des hôtes.

Hormis ces fonctions, ce panel reste assez limité :

- La fonction permettant de créer des utilisateurs donne immédiatement un contrôle total à un utilisateur autorisé. C'est-à-dire que si on donne des droits à un utilisateur pour démarrer et arrêter les conteneurs, il pourra arrêter absolument tous les conteneurs (même celui contenant Shipyard).
- Les données sur le site ne sont pas cachées. Pour peu que l'on puisse se loguer, on peut lire les résultats d'exécution de tous les conteneurs. Cela pose problème avec notre idée de confidentialité
- En effectuant des recherches sur le Github officiel du logiciel, on se rend bien vite compte que beaucoup de fonctionnalités en développement ne sont plus maintenues : C'est le cas, par exemple, de la fonctionnalité de Registry intégrée à Shipyard. Elle ne fonctionne plus qu'avec des versions de Docker Registry déclarée dépréciée[26]. Les utilisateurs de Shipyard réclament pourtant cette fonctionnalité, mais personne parmi la communauté des développeurs n'aura répondu à cet appel en suspens depuis bientôt 10 mois[27].

Finalement, si Shipyard offre une interface visuelle bienvenue à un hôte Docker, il comporte bien trop de lacune pour être implémenté dans notre projet. Il reste cependant quelque chose d'utile que l'on pourrait laisser installer pour aider les administrateurs à surveiller efficacement le système. Il n'est cependant pas adapté pour des utilisateurs extérieurs.

---

[26] Dans l'informatique, une fonctionnalité ou une technologie est déclarée dépréciée quand son développeur estime qu'elle ne doit plus être utilisée.

[27] https://github.com/shipyard/shipyard/issues/625 - demande de support de la registry v2 sur Shipyard en suspens depuis octobre 2015 – accédé le 13.06.16

On peut également ajouter que la plateforme Kitematic[28], aujourd'hui en alpha, est destinée à accomplir le même travail dans le futur.

4.4.6 Web service en PHP

Afin d'être sûr de pouvoir fournir exactement ce qui est nécessaire aux clients, la solution la plus simple serait peut-être finalement de développer nous même un service. Nous allons pour se faire créer un web service *from scratch* en PHP.

Pour ce faire, nous pouvons à nouveau nous servir de la puissance de Docker ! Nous utiliserons encore une fois une image présente sur Docker Hub. Il s'agit cette fois de l'image officielle de PHP, utilisant un serveur Apache.

Nous allons donc reprendre cette image et y ajouter les commandes suivantes :

```
FROM php:5.6-apache

COPY src/ /var/www/html/
COPY config/php.ini /usr/local/etc/php/
```

Les copies servent respectivement à déplacer les sources à la racine du site et le fichier de configuration PHP. Ces fichiers sont présents aux côtés du Dockerfile. Il est important de signaler que PHP recommande de fournir un fichier php.ini manuellement à l'image. Ce fichier contient toute la configuration du compilateur du code PHP. Si on ne le fourni pas, il utilisera un modèle par défaut. Cela pourrait entrainer des problèmes dans le futur. Il s'agit donc d'une pratique recommandée.

Le conteneur de cette image peut ensuite être construit et lancé simplement. J'ai décidé de lier mon port 80 à ce conteneur, car il s'agit du port http standard. Les web services fonctionnant via ce protocole.

Le service est créé selon la définition standard définie sur le site de PHP. Il peut être appelé d'un simple http GET sur l'API.

```
http://serveurURL/api.php?action=launchContainer&image=monImage
```

L'appel à Docker, en revanche, est plus complexe. En effet, il doit supporter les http POST et le passage de paramètre complexe. La Remote API a besoin d'un fichier de paramètre Json pour pouvoir créer un nouveau conteneur.

Pour résoudre ce problème, nous avons utilisé un plugin intégré dans PHP : cURL.

cURL est une interface habituellement utilisée en ligne de commande permettant d'effectuer des requêtes http poussées. Nous avons ici utilisé son intégration dans PHP.

Le web service créé contient une fonction pouvant être appelé par l'extérieur. Elle permet de lancer l'équivalent d'un Docker run sur une image. Pour rappel, un Docker run effectue un Push, un Create, un Start, et un Logs. L'application PHP effectue donc toutes ces commandes à la suite puis retourne les logs en tant que résultat.

---

[28] Kitematic est une interface graphique pour Docker officiel. Plus d'information sur https://kitematic.com/ - accédé le 13.06.2016

4.5 Comparaison des méthodes de contrôle

Dans le chapitre précédent, nous avons trouvé et essayé 3 méthodes différentes pour contrôler nos conteneurs depuis un environnement distant. 2 d'entre eux sont des logiciels open source existant, le dernier a été développé par nos soins. Nous allons ici parler des avantages et des défauts de chaque plateforme et de celle qui sera la plus intéressante à implémenter.

4.5.1 Complexité de Jenkins

Après l'installation et les tests sur Jenkins, nous pouvons nous rendre compte qu'elle offre énormément de possibilités pour gérer les projets internes. Or, notre objectif principal est de rendre l'infrastructure accessible à des personnes extérieurs.

Malgré nos plugins installés, il sera toujours possible pour une personne de lancer des processus Jenkins ou d'essayer de lancer quelque chose d'autres grâce aux nombreux outils natifs que l'on ne peut masquer. De plus, les utilisateurs peu avertis risquent d'avoir de la difficulté à se déplacer sur la plateforme.

En bref, Jenkins est une plateforme très complète, mais dont le but est de fournir un outil de CI à des utilisateurs avertis et faisant partie de l'entreprise. Pour une utilisation extérieure, cette solution n'est pas la plus optimale.

4.5.2 Lacunes de Shipyard

Shipyard apporte une grande valeur sur un serveur Docker : Une interface graphique. Malheureusement, la gestion des droits sur ce logiciel est elle aussi très minimaliste. La plateforme se veut comme étant une interface pour Docker Swarm, et pas comme une télécommande distante.

De plus, comme relevé plus haut pas ce rapport, le manque de soutien de la communauté pour ce projet est décourageant. Il peut s'avérer risquer d'investir dans une technologie qui pourrait être abandonnée.

Shipyard ne répond pas à nos attentes et doit être écarté.

4.5.3 Adaptabilité des services web

Nous avons pu essayer et voire qu'utiliser un service web développé par nos soins reste la solution la plus souple. Il est possible de créer uniquement ce que l'on a besoin. Cependant, le temps de développement est également plus important. Avec cette solution, on ne jouit pas de l'entretient qu'une communauté apporte à son logiciel. On est plus responsable de ce que l'on crée.

En utilisant un conteneur avec Apache, notre solution développée en PHP garde les avantages d'un système encapsulé. Il est très facile de déplacer ou de modifier l'exécution, voire de la copier pour effectuer des tests sans déranger le système productif.

4.5.4 Synthèse

Après la mise en place de toute ces solutions, une d'entre elle sort du lot : Celle utilisant un service web. C'est la seule qui permet réellement de fournir ce dont nous avons besoin et rien de plus. Elle demande en revanche plus de temps d'implémentation et d'entretiens.

Nous recommandons donc d'utiliser des web services dans le système productif. Il est cependant également possible d'implémenter plusieurs solutions et d'utiliser celle qui correspond le plus au besoin du moment.

Dans la suite du travail, nous déploierons sur le serveur la solution utilisant les web services.

4.6 DÉPLOIEMENT

Cette section du document décrit comment Docker doit être installé sur un serveur Ubuntu accessible uniquement via SSH. Nous allons également aborder quelques points plus complexes, comme la sécurisation des données.

Lorsque l'on travaille dans un domaine accessible depuis internet, il est important de réfléchir aux autorisations et à ce que pourrait faire une personne tierce mal intentionnée.

Nous pouvons également dire que cette partie va réutiliser les éléments déjà vu dans les parties précédentes du rapport. Nous n'allons donc pas réexpliquer tous les détails vus pendant les recherches.

Toutes les parties techniques réalisée l'ont été sur la machine *vlhrbbigmed1.hevs.ch*

Le Docker Remote API est déjà ouvert sur le port 2375 du localhost.

4.6.1 Installation de Docker

Suivant le tutoriel trouvé sur le site de Docker[29], l'installation de Docker sur Ubuntu est facile.

```
$ apt-get install docker-engine
$ service docker start
```

Si vous désirez tester l'installation, il vous suffit ensuite de lancer une image se trouvant sur Docker Hub. Docker recommande l'utilisation de la machine « hello-world » à ces fins.

Nous allons maintenant aborder quelque chose de nouveau : L'installation de Docker Compose. Ce composant était installé par défaut sur Docker Toolbox. Ce n'est pas le cas sur Linux. Nous allons donc devoir l'installer.

Docker compose étant écrit en Python[30], nous allons utiliser le gestionnaire de package de ce langage :

```
$ apt-get -y install python-pip
$ pip install --upgrade pip
```

Puis :

```
$ pip install docker-compose
```

Pour tester l'installation :

```
$ docker-compose –version
```

```
root@vlhrbbigmed1:~# docker-compose --version
docker-compose version 1.7.1, build 6c29830
root@vlhrbbigmed1:~#
```

Figure 28 : Installation Compose réussie

Docker est maintenant installé.

[29] https://docs.docker.com/engine/installation/linux/ubuntulinux/ - Docker Docs – accédé le 06.07.16
[30] https://docs.docker.com/compose/install/ - Docker Docs – accédé le 06.07.16

4.6.2 Registry sécurisée

Maintenant que nous travaillons sur un serveur déployé, nous ne pouvons plus utiliser simplement la registry vue dans le chapitre 4.3 seule. Nous avons besoin de lui ajouter une couche supplémentaire : La sécurité.

Cette sécurité doit être implémentée dans un deuxième conteneur. Celui-ci s'occupera des connexions sécurisées vers l'extérieure et de l'authentification des utilisateurs.

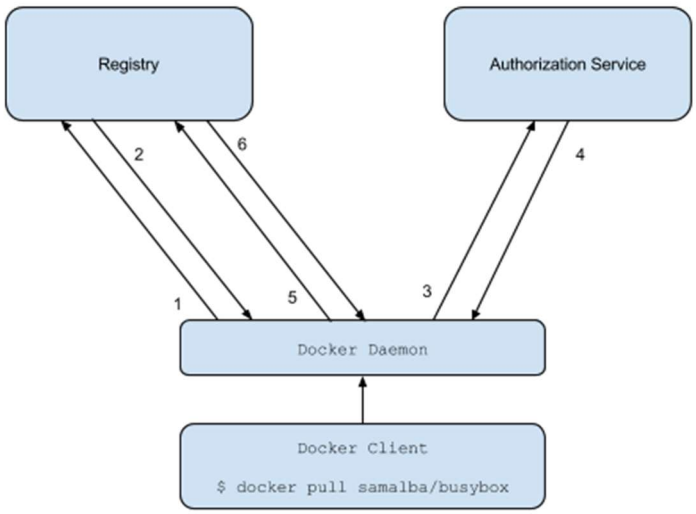


Figure 29 : Fonctionnement Docker autorisation[31]

Toute cette partie du déploiement s'inspire de cet excellent tutoriel[32] par Nik van der Ploeg récupéré sur www.digitalocean.com.

L'image Registry est en tout point semblable à celle que nous avons utilisée précédemment.

La partie authentification sera elle gérée par un service Nginx. Nginx est en effet très efficace et populaire dans la création de proxy.

[31] https://docs.docker.com/registry/spec/auth/token/ - Docker Docs – accédé le 05.07.2016

[32] https://www.digitalocean.com/community/tutorials/how-to-set-up-a-private-docker-registry-on-ubuntu-14-04 - tutoriel digitalocean - accédé le 06.07.16

Nous allons utiliser Docker Compose pour gérer les deux images simultanément. Voici le fichier de description docker-compose.yml :

```
nginx:
  image: "nginx:1.9"
  ports:
    - 443:443
  links:
    - registry:registry
  volumes:
    - ./nginx/:/etc/nginx/conf.d:ro
registry:
  image: registry:2
  ports:
    - 127.0.0.1:5000:5000
  environment:
    REGISTRY_STORAGE_FILESYSTEM_ROOTDIRECTORY: /data
  volumes:
    - ./data:/data
```

Le serveur Nginx, lui, est configuré pour filtrer les flux entrant suivant une authentification utilisateur. Il utilise des certificats provenant de Let's Encrypt pour activer l'encryptage SSL. La création de nouveaux utilisateurs peut être géré via un fichier au format Apache. On utilise pour les gérer l'utilitaire htpasswd[33] dans le répertoire de Nginx.

Nous pouvons maintenant pousser des images dans la registry beaucoup plus facilement.

```
vlhrbbigmed1.hevs.ch/[image]
```

Nous n'avons plus besoin de spécifier le numéro de port. Celui du protocole https est utilisé par défaut.

Avant d'interagir avec ce système, le client doit commencer par connecter son Docker Daemon en utilisant la commande :

```
$ docker login vlhrbbigmed1.hevs.ch
```

Son système va ensuite lui demander de se connecter à l'aide d'information de connexion qui doivent exister sur le serveur[34].

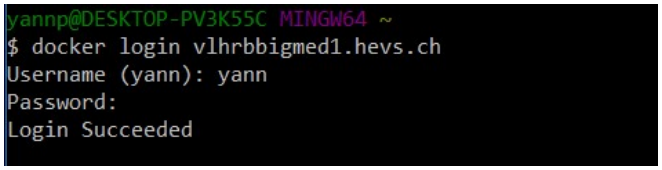


Figure 30 : Docker login réussi

[33] Pour plus d'information sur cet utilitaire, référez-vous à sa documentation officielle https://httpd.apache.org/docs/current/fr/programs/htpasswd.html - accédé le 12.07.16

[34] Il s'agit des comptes crée avec l'utilitaire htpasswd

Voici également les commandes pour relancer les conteneurs en cas d'arrêt :

```
$ cd /docker-registry
$ docker-compose up
```

Dans un système Docker, les conteneurs sont supprimés lorsque le système s'arrête. Cet évènement peut se produire même dans un système productif, c'est pour cela qu'il va être important d'y être attentif.

Toutes les données sont stockées dans des volumes. Ainsi, les configurations de Nginx et les données placées dans la registry sont sauvée sur le disque de l'hôte. Nous n'avons donc pas à craindre les coupures de ce côté-là.

Par contre, il n'existe aucun automatisme capable de relancer une composition de conteneur dans Docker. C'est pour cela que je recommande d'écrire un script de démarrage tel que Upstart ou Systemd qui va relancer automatiquement les conteneurs après un redémarrage machine.

4.6.3 Méthode de contrôle

Nous allons maintenant chercher à implémenter une méthode de contrôle vue précédemment sur notre serveur.

Cependant, une complication s'installe : Un service d'authentification existe à présent. Nous pouvons d'ores et déjà écarter les deux solutions utilisant des logiciels tierces car ils ne permettent pas de gérer le login à une registry. Une proposition qui serait implémentable facilement serait d'ouvrir un domaine sans authentification sur le proxy. Ce domaine ne serait ensuite accessible que par ces plateformes. Nous partirions dans ce cas de l'hypothèse que tout utilisateur connecté à ces logiciels sait ce qu'il fait et qu'on peut lui faire confiance.

Le cas le plus intéressant reste cependant de développer notre web service. Dans ce cas, nous appelons nous même l'API de Docker. Il conviendra donc d'ajouter les fonctionnalités d'authentification au code PHP qui a été écrit précédemment.

Pour ce faire, il faut ajouter le nom d'utilisateur et le mot de passe crypté en base64 dans un header du POST de l'appel CURL au pull. Pour être plus simple, voici le code PHP supplémentaire dans le web service :

```php
$data = array(
"username"=> "yann",
"password"=> "yann"
);
$data_string = json_encode($data,true);
$headauth = array("X-Registry-Auth:".base64_encode($data_string));
curl_setopt($session, CURLOPT_HTTPHEADER, $headauth);

// Tell curl not to return headers, but do return the response
curl_setopt($session, CURLOPT_HEADER, false);
curl_setopt($session, CURLOPT_RETURNTRANSFER, true);
```

Figure 31 : Code PHP authentification

Comme on peut le voir, il s'agit uniquement d'ajouter le header « X-Registry-Auth ».

Ensuite, nous pouvons déplacer notre service sur le serveur. Je recommande pour se faire de pousser l'image contenant Apache dans la Registry, puis de se connecter au serveur par SSH et d'effectuer le run.

```
$ sudo docker run -d --net=host vlhrbbigmed1.hevs.ch/webservice
```

Une chose nouvelle également est l'argument net. Ici, on informe Docker de fusionner le réseau du conteneur avec celui de la carte réseau locale. En effet, l'API Docker doit être protégée sur le serveur, aussi doit-on la rendre accessible uniquement au localhost. Cette option devient donc nécessaire.

4.7 Performances de Docker contre les VMs

Avec le développement des nouvelles technologies informatiques, les entreprises sont toujours à la recherche de la solution qui pourra offrir le plus en coutant le moins. Les promesses de Docker sont importantes car il ambitionne de révolutionner l'architecture des centres de données modernes. Peut-on pourtant dire que les conteneurs à la Docker sont plus performant que l'architecture avec machines virtuelles actuellement implémentée dans la plupart des centres ?

Premièrement, les conteneurs utilisent les ressources même de la machine. On peut donc constater une nette amélioration. Une étude d'IBM sur la comparaison entre les machines virtuelles et les conteneurs a démontré grâce à des benchmarks que les performances d'un conteneur étaient très proches de celles offertes par le système natif [6].

De plus, sur un serveur exécutant Docker, les conteneurs vont toujours occuper toute la puissance disponible comme un OS le ferait. Avec les machines virtuelles, elles sont limitées par la configuration de l'enveloppe, mais également par les limitations du système client virtualisé.

Mais, d'après la même étude d'IBM, les conteneurs obtiennent un plus mauvais résultat sur les performances réseaux lorsque « Docker nat » est utilisé [6]. Il s'agit de la couche réseau supplémentaire permettant de mapper un port entre le conteneur et l'hôte[35]. Si un conteneur a besoin de beaucoup de performances réseaux, il est cependant possible d'éviter cette couche grâce à la fusion entre l'interface virtuelle et hôte[36].

Encore, la virtualisation est une technologie fortement implantée, ayant fait ses preuves [2]. On ne peut pas en dire autant sur les technologies à base de conteneurs. Docker en tant que tel existe seulement depuis 2013[37]. On peut donc dire qu'une installation à base de VM représente une certaine sécurité de mise en place.

---

[35] Paramétrer avec l'option -p dans le terminal docker. Tel que « Docker run -p 80 :80 apache »

[36] En remplaçant l'argument -p par « --net=host »

[37] http://venturebeat.com/2016/01/22/dotcloud-the-cloud-service-that-gave-birth-to-docker-is-shutting-down-on-february-29/ - article sur la fermeture de dotCloud, l'entreprise qui a donné naissance à Docker – accédé le 28.06.2016

Finalement, d'un point de vue des performances pures, Docker est imbattable. Comme le montre le graphique suivant, récupéré dans l'étude d'IBM, les « performances pures » du processeur sont bien meilleures dans un système conteneurisé.

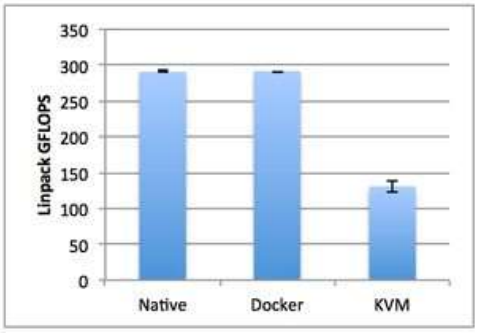


Figure 1. Linpack performance on two sockets (16 cores). Each data point is the arithmetic mean obtained from ten runs. Error bars indicate the standard deviation obtained over all runs.

Figure 32 : Benchmark IBM Docker vs KVM[38][39]

Nous pouvons donc constater la valeur ajoutée que ce genre de système apporte dans un centre de calcul. Les promesses de Docker sont grandes et elles semblent tenir la route.

[38] KVM est un hyperviseur libre utilisé sous Linux

[39] http://domino.research.ibm.com/library/cyberdig.nsf/papers/0929052195DD819C85257D2300681E7B/$File/rc25482.pdf - étude IBM. Crédit : IBM – accédé le 29.06.2016

4.8 DOCKER ET LE GPU COMPUTING

Le GPU computing est un enjeu majeur dans le domaine de la science des données moderne. Plusieurs études ont démontré que les cartes destinées au calcul graphique possédaient le potentiel d'accélérer grandement le temps de calcul dans un Datacenter[40].

D'après le fondeur NVidia, les cartes graphiques possédant une quantité bien plus importante de cœur que les processeurs, sont beaucoup plus « douées » pour exécuter les petites tâches.

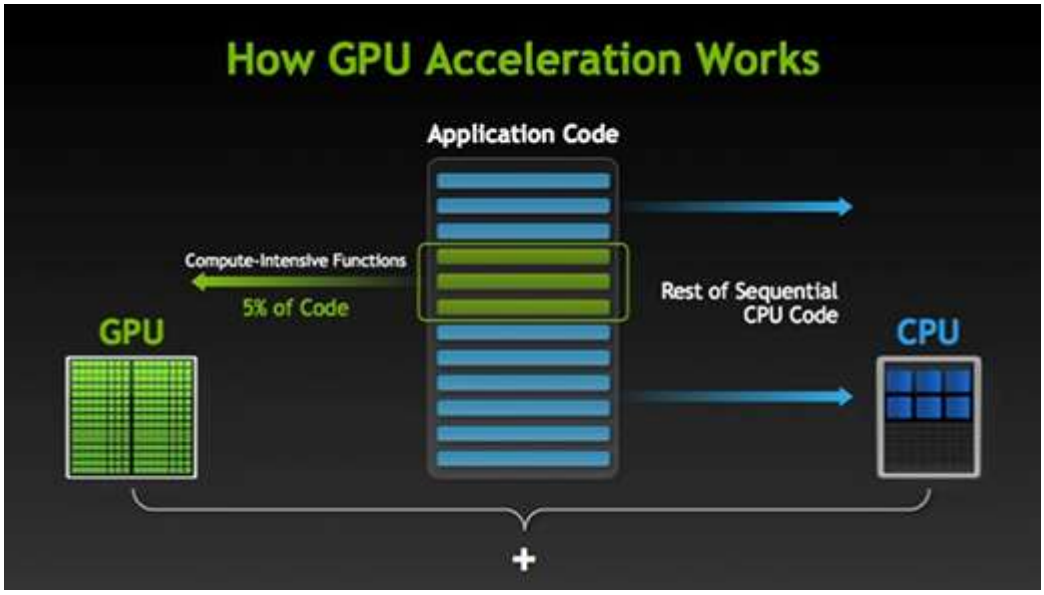


Figure 33 : Comment le GPU computing fonctionne[41]

Pour faciliter l'usage de Datacenter avec le GPU computing, NVidia propose une solution de calcul en parallèle nommée CUDA[42]. CUDA permet le calcul en parallèle entre un ou plusieurs CPU(s) et GPU(s). Le nombre d'applications compatibles avec cette technologie est en augmentation. Une implémentation est même disponible avec Docker.

[40] D'après plusieurs sources officielles de NVidia, fondeur et fabriquant de carte graphique

[41] http://www.nvidia.com/object/what-is-gpu-computing.html - site de NVidia. Crédit : Nvidia.com – accédé le 13.06.16

[42] http://www.nvidia.fr/object/cuda-parallel-computing-fr.html - présentation de CUDA, site de NVidia – accédé le 13.06.16

Elle se présente sous la forme d'une série d'images préconfigurées par NVidia pour fonctionner avec CUDA. Pour installer ce système, il faudra bien faire attention aux drivers des GPU. Il faudra ensuite installer un plugin sur l'hôte qui pourra ensuite exécuter toute une série de commandes supplémentaires via « NVidia-docker ».

L'architecture NVidia-Docker se présente comme une implémentation de Docker classique. La principale différence se situant au niveau des pilotes graphiques de l'hôte.

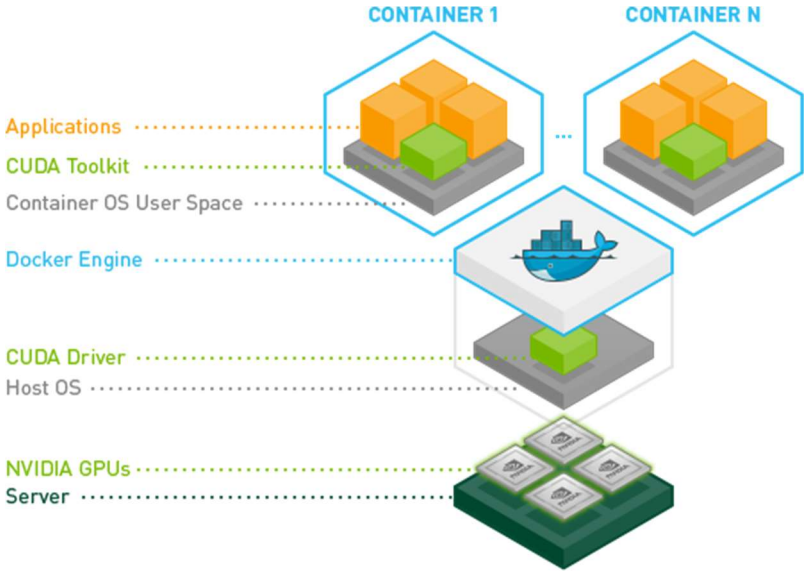


Figure 34 : Architecture NVidia Docker[43]

En conclusion, NVidia CUDA pour Docker apporte les avantages du GPU computing à une architecture mettant en place les conteneurs en tant que service.

[43] https://github.com/NVIDIA/nvidia-docker - documentation du projet Nvidia Docker – accédé le 13.06.16

4.9 RECYCLAGE DES IMAGES

Après qu'un conteneur ait effectué son travail, il conviendra de s'en débarrasser afin de libérer de l'espace mémoire. Nous pouvons aisément supprimer une image ou un conteneur avec les commandes :

```
$ Docker rm xxx          # pour un conteneur
$ docker rmi xxx         # pour une image
```

Il ne serait également pas compliqué d'ajouter un appel à notre web service PHP pour qu'il fasse le ménage après le retour des logs.

Là où ça se complique, c'est dans le nettoyage de la registry. En effet, aucune fonction n'est prévue pour supprimer du contenu à l'intérieur. Ceci est contraire à la philosophie d'un système de contrôle de version tel qu'elle.

Cela n'est pas si grave, car elle est capable d'effectuer un delta et de ne garder qu'une version des couches strictement identiques. Cela veut dire que si 5 images exécutant du code sous Ubuntu sont poussées, la registry gardera une image Ubuntu, et les 5 couches avec le code. L'espace mémoire utilisé est ainsi réduit. Les couches les plus volumineuses sont celles qui ont le plus de chances d'être présente plus fréquemment.

Si l'on désirerait tout de même effectuer un nettoyage de la registry, il est possible de supprimer directement les dossiers de référencement dans le volume lié. Celui-ci correspond en tout point au contenu. La communauté Docker[44] souhaitant effectuer ce recyclage a pris l'habitude d'écrire des scripts qui vont supprimer ces dossiers.

[44] https://github.com/docker/docker-registry/labels/delete - Github de la registry – sujet de la communauté traitant du nettoyage – accédé le 14.07.16

4.10 Sécurité de l'environnement

En permettant à des utilisateurs extérieurs d'envoyer des conteneurs sur le réseau interne, nous créons une éventuelle possibilité de faire entrer des logiciels malicieux. En effet, notre processus ne scanne pas le contenu envoyé. Nous partons du principe que l'utilisateur sait ce qu'il envoie. Nous allons donc regarder ici quels sont les risques liés au contenu envoyé depuis l'extérieur.

Premièrement, et cela pourrait être vu comme un inconvénient vis-à-vis des machines virtuelles, une bonne partie des couches des conteneurs est partagée avec l'OS hôte. Cela veut dire qu'un attaquant qui a réussi à déplacer son image dans l'infrastructure atteindra beaucoup plus rapidement le kernel de l'hôte qu'avec une solution virtualisée [7]. Docker[45] est conscient de ce problème et fournit certaines recommandations.

Pour autant, les conteneurs offrent tout de même une isolation importante. Toutes les actions sont exécutées sur le kernel de l'hôte, mais dans des namespaces[46] différents. Cela veut dire que les exécutions de l'hôte et du conteneur sont isolées l'une de l'autre.

De plus, il est important de contrôler ce qu'un conteneur peut faire. Ainsi, le contrôle de Docker par web service proposé dans ce travail offre cette possibilité. Il faut, par défaut, interdire aux utilisateurs de créer des volumes ou de mapper des ports s'ils n'en ont pas besoin [7]. Si un projet le requiert, il faudra mettre en place une solution pour vérifier ce qui en sort.

Finalement, la sécurité est quelque chose à laquelle il faut être attentif. Dans le cadre de mon projet, et à l'aide d'articles[47] que j'ai lu, voici les quelques règles que j'implémenterai dans ce système :

- Toujours authentifier les utilisateurs qui ont un accès
- N'utiliser que des communications par https
- Maintenir le kernel de l'hôte à jour afin d'éviter les exploits provenant des conteneurs
- Toujours maintenir Docker à jour. Docker connait les risques liés aux conteneurs et travaille activement sur des solutions pour augmenter la sécurité.
- Maintenir le proxy et la registry à jour également

En appliquant ces règles, nous devrions maintenir un niveau de sécurité acceptable. Il ne faut pas oublier que Docker est une technologie encore jeune. Elle manque encore de robustesse.

---

[45] https://docs.docker.com/engine/security/security/ - Docker docs – accédé le 13.07.16

[46] Espace de nom. Représente un lieu abstrait dans l'informatique.

[47] https://www.oreilly.com/ideas/docker-security?intcmp=il-webops-free-article-lgen_five_security_concerns_when_using_docker et https://www.oreilly.com/ideas/five-security-concerns-when-using-docker principalement - accédé le 13.07.16

5 CONCLUSION

Nous voici arrivés au bout de ce projet. Nous avons pu constater la puissance de Docker dans la création de conteneur. Ceux-ci se trouvent souvent plus pratique que des machines virtuelles. Nous avons vu que créer un conteneur était très facile grâce aux nombreuses images disponible sur le Hub.

Nous avons ensuite exploré les possibilités de l'API REST intégrée dans Docker. Nous avons vu qu'il était possible de piloter facilement un hôte à distance. Nous avons profité de cette étape pour créer un environnement de test virtualisé.

C'est à ce moment-là que nous avons pu réellement commencé les parties concrètes du projet.

Nous avons débuté avec la création d'une registry. Celle-ci permet de transférer des images Docker entre les hôtes avec une facilité déconcertante. Nous avons vu que l'intégration de git dans Docker nous offrait cette facilité.

Nous avons ensuite effectué une recherche des méthodes de contrôle des conteneurs sur l'hôte distant. Pour ce faire, nous avons effectué une comparaison entre 2 plateformes et un service développé durant le projet. Nous avons pu voir que, malgré la puissance des plateformes existantes, elles ne répondent pas forcement au besoin de notre projet. C'est pour cela qu'une solution maison, développée ici en PHP, se démarque.

La solution développée en PHP se sert du plugin CURL pour envoyer des commandes à l'API vue plus tôt. Malgré une légère augmentation en complexité, nous avons une solution très souple. Nous avons pris la décision d'en faire un web service REST également pour laisser la liberté de créer plus tard un client.

Nous nous sommes ensuite attelés au déploiement de nos solutions sur une machine virtuelle se trouvant sur les serveurs de l'institut. Cette fois, nous nous sommes attardés davantage sur les principes de sécurité, et avons ajouté une authentification des utilisateurs qui tentent d'accéder la registry. Nous avons pour se faire ajouté un proxy inverse et un cryptage par certificat aux communications vers l'extérieur.

Nous avons immédiatement enchainé par l'adaptation de web service PHP pour y ajouter l'authentification et la sécurisation. Nous l'avons ensuite déployée sur le serveur afin de posséder une démo permettant de stocker puis de lancer des images de A à Z.

Nous avons ensuite terminé le travail avec quelques recherches supplémentaires. Nous avons vu que Docker offre des performances impressionnantes par rapport aux VMs, avec une perte de puissance de calcul minime. Nous avons ensuite exploré les possibilités existantes pour faire du GPU computing avec conteneur. Nous avons vu que Nvidia CUDA possède une intégration.

Ensuite, nous avons fait quelques tests par rapport au recyclage des images. Nous avons vu que tout n'est pas parfait. A nouveau, seul le web service offre la souplesse de faire exactement ce que l'on veut. Le point noir est dans la registry qui ne possède pas de commande de suppression. Cela n'est cependant pas si grave, car les couches les plus volumineuses ne se répètent pas. De plus, il est toujours possible de nettoyer directement le volume sur le disque.

Finalement, nous avons fait le point sur la sécurité du système exécutant Docker. Nous avons constaté qu'il s'agit là d'un inconvénient de Docker. Nos conclusions nous ont montré que Docker ne protège pas l'hôte contre un exploit envers son kernel. Les machines virtuelles se montrent plus résistantes à ce niveau-là.

Nous avons réussi à atteindre tous les objectifs cités au début du travail. La solution développée tient la route. Il convient cependant de fixer quelques règles et point auquel le gestionnaire doit être attentif.

En exécutant un conteneur dans l'infrastructure, l'utilisateur distant obtient un certain pouvoir. Il est possible qu'il puisse en abuser pour tromper la sécurité mise en place. Je recommande aux futures personnes qui désirent utiliser ce système de surveiller les accès, et d'isoler la machine hôte du reste de l'infrastructure. Il conviendra ensuite d'ouvrir uniquement ce qui doit l'être.

En soi, Docker possède encore des lacunes quant à l'isolation des conteneurs à l'hôte.

Voici une petite liste non exhaustive de ce qui pourrait encore être fait pour ce projet :

La solution plébiscitée par ce travail pour le contrôle est un web service écrit ici en PHP. Il serait intéressant de créer un client qui appelle les fonctions de ce web service pour fournir une interface utilisateur plus poussée.

Le web service lui-même fait également un peu office de proof-of-concept et pourrait être amélioré. Le travail décrit comment utiliser les options de Docker avec, mais ne le fait pas directement. Cela pourrait être complété pour créer un service qui réponde à tous les cas tel que :

- Les ports
- Les volumes
- Les variables d'environnements
- Etc.

Quelques fonctions managériales pourraient également être prévues :

- Un script de démarrage qui relance la composition de la registry et du proxy
- Un script capable de nettoyer le volume de la registry après un certain temps

Dans les perspectives de recherche futures, et afin de pousser le concept, je conseille de se concentrer sur l'aspect sécuritaire de notre système. C'est le point qui est encore malheureusement le plus flou avec Docker, et chaque nouvelle version apporte son lot de nouveautés dans le domaine.

Pour conclure, j'aimerais encore pouvoir parler de la version Windows de Docker qui est devenue disponible en beta vers la fin de mon temps de travail sur ce projet. Il est maintenant possible de créer des images utilisant Windows comme OS et de les exécuter nativement sur Windows 10. Je n'ai malheureusement pas eu le temps de me pencher sur la question. Cependant, cet ajout apporte énormément de richesses dans l'écosystème des conteneurs qui n'en devient que plus intéressant.

Docker est une technologie jeune qui apporte beaucoup de promesses et suscite énormément de mouvements dans le domaine du cloud. Les enjeux sont énormes et son utilisation va probablement grandir dans les années à venir.

6 REFERENCES

[1] H. Müller, J. Kalpathy–Cramer, A. Hanbury, K. Farahani, R. Sergeev, J. H. Paik, A. Klein, A. Criminisi, A. Trister, T. Norman, D. Kennedy, G. Srinivas, A. Mamonov et N. Preuss, «Report on the Cloud-Based Evaluation Approaches Workshop 2015,» 2015.

[2] K.-T. Seo, H.-S. Hwang, I.-Y. Moon, O.-Y. Kwon et B.-J. Kim, «Performance Comparison Analysis of Linux Container and Virtual Machine for Building Cloud,» 2014.

[3] A. Hanbury, H. Müller, K. Balog, T. Brodt, G. V. Cormack, I. Eggel, T. Gollub, F. Hopfgartner, J. Kalpathy-Cramer, N. Kando, A. Krithara, J. Lin, S. Mercer et M. Potthast, «Evaluation-as-a-Service: Overview and Outlook,» 2015.

[4] J.-P. GOUIGOUX, Docker - Prise en main et mise en pratique sur une architecture micro-services, Editions ENI, Octobre 2015.

[5] T. Bartolone, Red Hat Enterprise Linux - CentOS : mise en production et administration de serveurs, St-Herblain: Edition ENI, 2015.

[6] W. Felter, A. Ferreira, R. Rajamony et J. Rubio, «IBM Research Report : An Updated Performance Comparison of Virtual Machines and Linux Containers,» Austin Research Laboratory, Austin, 2014.

[7] A. Mouat, «Docker security Using containers safely in production,» 6 Octobre 2015. [En ligne]. Available: https://www.oreilly.com/ideas/docker-security?intcmp=il-webops-free-article-lgen_five_security_concerns_when_using_docker.

7 DECLARATION DE L'AUTEUR

Je déclare, par ce document, que j'ai effectué le travail de Bachelor ci-annexé seul, sans autre aide que celles dûment signalées dans les références, et que je n'ai utilisé que les sources expressément mentionnées. Je ne donnerai aucune copie de ce rapport à un tiers sans l'autorisation conjointe du RF et du professeur chargé du suivi du travail de Bachelor, y compris au partenaire de recherche appliquée avec lequel j'ai collaboré, à l'exception des personnes qui m'ont fourni les principales informations nécessaires à la rédaction de ce travail et que je cite ci-après : -.