Table des matières

Liste des figures

Liste des tableaux

Avant-propos

Ce travail est proposé par mon professeur responsable et il fait partie d'un projet en cours d'implémentation par une équipe de l'IIG.

Le but est donc de se baser sur l'existant et fournir des analyses complémentaires, ainsi qu'un prototype (*proof-of-concept*) qui illustre les tests réalisés et les résultats qui en découlent. Ces éléments peuvent ensuite être inclus dans le projet de l'IIG.

J'ai choisi ce thème, car j'éprouve un grand intérêt en ce qui concerne l'informatique médicale, la configuration de systèmes, la protection des données et la sécurité. De plus, j'ai au préalable beaucoup entendu parler de Docker et je veux en apprendre d'avantage sur cette technologie et son principe d'utilisation.

Ce rapport est basé sur la norme de mise en forme de l'*American Psychological Association* (APA), 6^{e} édition.

Remerciements

Je tiens à remercier toutes les personnes qui m'ont apporté leur aide et soutien dans le cadre de ma formation et de ce travail et en particulier mon professeur, **Dr. Henning Müller**, pour le temps qu'il a consacré à me suivre, me conseiller et me corriger, ainsi que **M. Ivan Eggel**, chercheur et développeur à l'Institut Informatique de Gestion de la HES-SO Valais, pour son précieux suivi technique du projet.

J'adresse aussi un très grand remerciement à mes proches pour leur soutien et la relecture de ce document.

Liste des abréviations

IIG	Institut Informatique de Gestion (HES-SO Valais)
OS	Operating system (Système d'exploitation)
API	Application Programming Interface
REST	Representational State Transfer (Type architecture Web)
LDAP	Lightweight Directory Access Protocol
AD	Active Directory (Services annuaire de Microsoft)
CPU	Central Processing Unit (processeur)
YAML	YAML Ain't Markup Language (langage de programmation)
HTTP	HyperText Transfer Protocol
LPrD	Loi sur la Protection des Données Personnelles (Vaud)
LPD	Loi fédérale sur la Protection des Données (Suisse)
P-LPD	Projet de révision de la LPD (Suisse)
RGPD	Règlement Général pour la Protection des Données (Europe)
CNIL	Commission Nationale de l'Informatique et des Libertés (France, Europe)

Glossaire

Dockerfile : fichier qui contient les commandes qui seront exécutées lors de la création d'une image Docker

Deep learning : *« Le deep learning ou apprentissage profond est un type d'intelligence artificielle dérivé du machine learning (apprentissage automatique) où la machine est capable d'apprendre par elle-même, contrairement à la programmation où elle se contente d'exécuter à la lettre des règles prédéterminées. »*[1]

Cluster : Groupe de machines (serveurs) indépendantes qui sont connectées à travers un réseau spécifique afin de fonctionner en tant que ressource « unique » centralisée.[2]

Product Backlog : Terme provenant de la méthodologie Agile, fichier contenant les fonctionnalités / composantes d'un projet souhaitées par le client (utilisateur final)

Hacker : Individu dont le but est de mettre à profit des compétences, des principes ou des logiciels afin de pénétrer dans une infrastructure informatique et d'y dérober, dans notre cas, des informations

---

1 https://www.futura-sciences.com/tech/definitions/intelligence-artificielle-deep-learning-17262/, consulté le 15.10.2018

2 http://www.businessdictionary.com/definition/cluster.html, consulté le 23.10.2018

1. Introduction

1.1. Contexte

Le contexte ci-dessous est basé sur le rapport d'Ivan Eggel *et al.* (2018).

De nos jours, la quantité de données collectées et les possibilités offertes par la technologie pour leur analyse (par exemple l'intelligence artificielle, les algorithmes, le *deep learning*, le matériel) augmentent de façon exponentielle, comme dans le secteur médical où l'informatique joue désormais un rôle conséquent.

Or, le partage de ces données peut être réalisé en envoyant des disques durs par voie postale ou via Internet, ce qui nécessite de larges bandes passantes et cela peut prendre un temps considérable de téléchargement. De plus, la protection des données personnelles et sensibles implique que les données médicales (dans notre cas) soient anonymisées ou pseudonymisées. Le consentement du patient doit être recueilli et des conditions de traitement doivent être approuvés par les chercheurs, ce qui implique des procédures administratives et peut mener à un manque de données dans le cadre d'une recherche.

C'est pourquoi un projet réalisé à l'IIG propose une solution qui permet un traitement plus rapide, léger et adaptable de données personnelles et sensibles, stockées en interne. Uniquement l'image Docker créée par un chercheur, contenant le code d'analyse, a accès aux données et celui-ci ne voit que le résultat de l'analyse.

L'architecture souhaitée est représentée par la figure 1 ci-dessous :

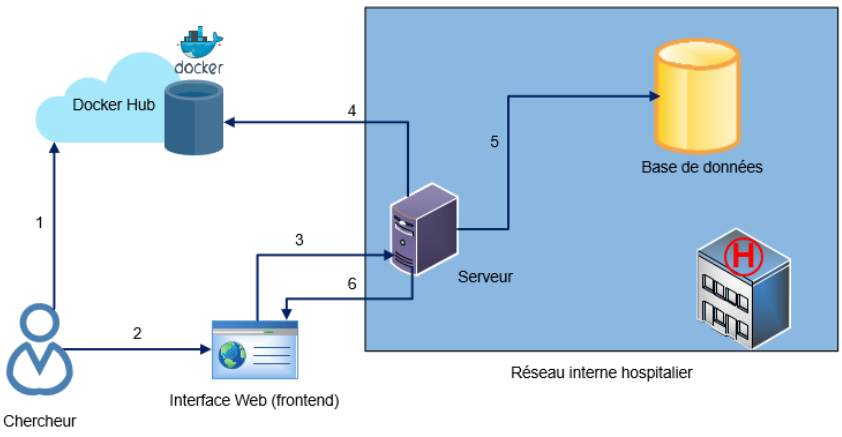


***Figure 1 Architecture du système à implémenter**. (1) Création de l'image Docker avec le code d'analyse et envoi sur le dépôt Docker public ; (2 et 3) Lancement d'exécution d'analyse à travers une interface Web ; (4) récupération de l'image concernée sur le dépôt par le serveur interne dans le réseau hospitalier ; (5) Exécution du*

code à l'intérieur de cette image en accédant aux données concernées ; (6) Affichage du résultat sur l'interface Web pour le chercheur.

Les membres de l'IIG ont déjà mis en place les éléments suivants : l'installation de *Docker Swarm* en local, la création d'une interface web pour lancer des conteneurs Docker liés avec des données internes, le choix d'environnement d'exécution (haute mémoire, haute capacité de processeur et besoins graphiques), le stockage des résultats du code d'analyse de données et l'évaluation de ces derniers.

Le but de ce travail est donc d'apporter des compléments qui peuvent ensuite être repris dans l'implémentation de ce projet de l'IIG.

1.2. Objectifs / étapes

Ce travail regroupe plusieurs objectifs. Nous réalisons d'abord une analyse de Docker (concepts, versions, standards, outils) et des risques liés à l'exécution de code étranger arbitraire. Ensuite, nous mettons en place un environnement local (Docker, *Swarm*, service web, interface web cliente). Enfin, nous analysons et testons la définition de quotas (mémoire et processeur), le blocage du trafic réseau qui n'est pas nécessaire au bon fonctionnement du *Swarm* et d'un conteneur, et le système de mise en attente (queue) des conteneurs.

En fonction du temps restant, nous fixerons une durée de vie à un conteneur, nous trouverons des alternatives à Docker Hub, nous mettrons en place un registre d'images privé et nous établirons une procédure de « Conditions d'utilisation » des données. Pour plus de détails concernant les objectifs, veuillez vous référer au cahier des charges (Annexe I).

Pour finir, nous établissons un bilan technique et une conclusion générale avec des améliorations potentielles du projet.

2. Etat de l'art de Docker

Docker est une plateforme Open Source mise en ligne pour la première fois en 2013. Elle utilise des conteneurs pour permettre le déploiement et l'exécution d'applications dans les Cloud d'entreprises ou publics. Nous présentons ci-dessous les principes, les versions, les standards, les outils et méthodes disponibles à travers Docker.

2.1. Principes de Docker

2.1.1. Containers

Docker est basé sur un fonctionnement en *containers* (ou conteneurs en français). Ce terme provient des conteneurs utilisés dans le transport de marchandises. Un conteneur représente *« une instance exécutable d'une Docker® image* […] *Chaque conteneur est une plateforme d'applications indépendante et sécurisée à partir de laquelle on peut accéder à des bases de données ainsi qu'à des ressources présentes dans un autre conteneur. »* (K. Laghzaoui, 2017).

Figure 2 Cargo containers [3]

Un conteneur peut être démarré, arrêté, déplacé ou supprimé via l'outil Docker en ligne de commande ou à travers un API.

2.1.1.1. Machines virtuelles vs. containers

Dans une machine virtuelle, un OS entier se charge, avec des processus et plusieurs applications qui peuvent n'avoir aucune utilité selon l'utilisation de la machine. Au contraire,

[3] https://pixabay.com/fr/d-affaires-des-conteneurs-de-fret-1845350/, récupéré le 16.10.2018

un conteneur Docker utilise uniquement les éléments nécessaires, définis dans l'image qu'il contient. Les ressources peuvent être partagées entre conteneurs.

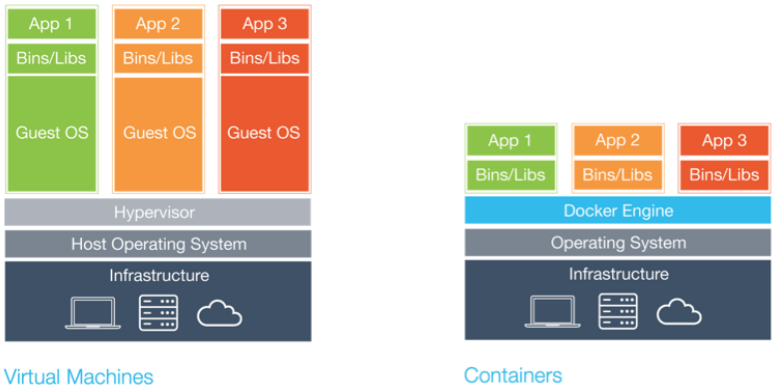


Figure 3 Comparaison VM et conteneurs[4]

Un fonctionnement par conteneurs présente des avantages par rapport à une machine virtuelle, comme indiqué dans le tableau ci-dessous (K. Laghzaoui, 2017).

[4] https://blog.docker.com/2015/08/docker-demo-faq/, récupérée le 18.10.2018

Caractéristiques	Machines virtuelles	Conteneurs
Système d'exploitation invité	Chaque machine virtuelle est exécutée directement sur le matériel	Tous les systèmes d'exploitation invités partagent le même système d'exploitation hôte et le même noyau Linux™
Sécurité	Plus sécurisé car chaque machine virtuelle a son propre noyau de système d'exploitation	Vulnérable à cause du partage du noyau Linux™ par les différents conteneurs
Performance	Performances limitées à cause du temps de latence entre le système d'exploitation invité et hôte	Performances proches du système d'exploitation hôte
Isolation	Partage de bibliothèques et de fichiers entre les invités et l'hôte impossible	Les conteneurs sont isolés mais peuvent partager un système d'exploitation, des bibliothèques ou des fichiers.
Temps de démarrage	Quelques minutes	Quelques secondes
Stockage	Beaucoup de mémoire car tout le système d'exploitation et ses programmes sont installés et exécutés	Les conteneurs prennent peu de mémoire car le système d'exploitation hôte est partagé
Déploiement	Lourd et difficile	Léger et facile

Tableau 1 Caractéristiques VM vs. conteneur

2.1.2. Docker Images

2.1.2.1. Fonctionnement

Comme mentionné dans le point 2.1.1, un conteneur Docker est constitué d'une image avec un identifiant unique. Cette image contient l'application, les librairies et tout autre fichier nécessaire au bon fonctionnement, tous accessibles uniquement en lecture.

Elle est composée d'une image de base à laquelle nous apportons des couches supplémentaires (images intermédiaires) en effectuant des modifications souhaitées via des commandes écrites dans un fichier appelé Dockerfile. Pour illustrer ceci, le Dockerfile en figure 3 sert à créer une image pour une application web NodeJS. Lors de la construction de l'image (création du conteneur), chaque commande est exécutée l'une après l'autre (Figure 4) et lorsque l'image est prête, nous pouvons afficher les couches (Figure 5). [5]

[5] https://medium.com/@jessgreb01/digging-into-docker-layers-c22f948ed612, consulté le 17.10.2018

```
FROM node:argon

# Create app directory
RUN mkdir -p /usr/src/app
WORKDIR /usr/src/app

# Install app dependencies
COPY package.json /usr/src/app/
RUN npm install

# Bundle app source
COPY . /usr/src/app

EXPOSE 8080
CMD [ "npm", "start" ]
```

Figure 4 Exemple Dockerfile

```
$ docker build -t expressweb .
Step 1 : FROM node:argon
argon: Pulling from library/node...
...
Status: Downloaded newer image for node:argon
 ---> 530c750a346e
Step 2 : RUN mkdir -p /usr/src/app
 ---> Running in 5090fde23e44
 ---> 7184cc184ef8
Removing intermediate container 5090fde23e44
Step 3 : WORKDIR /usr/src/app
 ---> Running in 2987746b5fba
 ---> 86c81d89b023
Removing intermediate container 2987746b5fba
Step 4 : COPY package.json /usr/src/app/
 ---> 334d93a151ee
Removing intermediate container a678c817e467
Step 5 : RUN npm install
 ---> Running in 31ee9721cccb
 ---> ecf7275feff3
Removing intermediate container 31ee9721cccb
Step 6 : COPY . /usr/src/app
 ---> 995a21532fce
Removing intermediate container a3b7591bf46d
Step 7 : EXPOSE 8080
 ---> Running in fddb8afb98d7
 ---> e9539311a23e
Removing intermediate container fddb8afb98d7
Step 8 : CMD npm start
 ---> Running in a262fd016da6
 ---> fdd93d9c2c60
Removing intermediate container a262fd016da6
Successfully built fdd93d9c2c60
```

Figure 5 Etapes création de conteneur (couches)

```
docker history <image>

$ docker history expressweb
IMAGE         CREATED    CREATED BY                       SIZE
fdd93d9c2c60  2 days ago /bin/sh -c CMD ["npm" "start"]   0 B
e9539311a23e  2 days ago /bin/sh -c EXPOSE 8080/tcp       0 B
995a21532fce  2 days ago /bin/sh -c COPY dir:50ab47bff7   760 B
ecf7275feff3  2 days ago /bin/sh -c npm install           3.439 MB
334d93a151ee  2 days ago /bin/sh -c COPY file:551095e67   265 B
86c81d89b023  2 days ago /bin/sh -c WORKDIR /usr/src/app  0 B
7184cc184ef8  2 days ago /bin/sh -c mkdir -p /usr/src/app 0 B
530c750a346e  2 days ago /bin/sh -c CMD ["node"]          0 B
```

Figure 6 Exemple affichage de couches pour une image

Une liste d'instructions qui peuvent être écrites dans un *Dockerfile* pour configurer une image se situe en annexe.

2.1.3. Services Docker

Les services Docker permettent d'échelonner les conteneurs et les distribuer sur plusieurs machines dans un *Swarm* (cf. chapitre Docker Swarm) sur lesquelles le processus dockerd est démarré. Les services peuvent être paramétrés à travers un fichier *Docker Compose* (cf. Docker Compose).

2.1.4. Docker Engine

Docker Engine est l'application de type client-serveur et le cœur de l'architecture Docker. Il gère les images, conteneurs, réseaux et volumes Docker. Il est composé principalement du processus « serveur » (*daemon*) *dockerd*, d'une API REST qui sert de lien entre les interfaces que les programmes peuvent utiliser et *dockerd*, et d'un client Docker en ligne de commande.[6]

2.1.5. Registres d'images Docker

Les images Docker peuvent être enregistrées dans un registre (*registry* en anglais) privé ou public.

2.2. Versions de Docker

Comme indiqué par J. Chelladhurai *et al.* (2017), Docker propose deux versions de plateforme : *Community Edition*, pour les développeurs et les acteurs opératifs, et l'*Enterprise Edition* pour les applications critiques et les partenaires qui apportent de la valeur ajoutée à Docker. Nous présentons les deux versions, mais nous retenons seulement la *Community Edition* dans la réalisation pratique de ce travail, étant donné sa gratuité et que les fonctionnalités qu'elle offre sont suffisantes dans le cadre de ce projet de l'IIG.

[6] https://docs.docker.com/engine/docker-overview/#docker-engine, consulté le 19.10.2018

2.2.1. Community Edition

Docker *Community Edition* (ou Docker CE) peut-être installé sur [7]:

- MacOS Desktop (macOS El Capitan 10.11 et supérieur)
- Windows 10 Pro Desktop
- Linux CentOS 7 ou supérieur
- Debian
  - Buster 10
  - Stretch 9 / Raspbian Stretch
  - Jessie 8 (LTS) / Raspbian Jessie
  - Wheezy 7.7 (LTS)
- Fedora 26 à 28 (64-bit)
- Ubuntu
  - Bionic 18.04 LTS
  - Xenial 16.04 LTS
  - Trusty 14.04 LTS

DESKTOP

Platform	x86_64
Docker for Mac (macOS)	✓
Docker for Windows (Microsoft Windows 10)	✓

SERVER

Platform	x86_64 / amd64	ARM	ARM64 / AARCH64	IBM Power (ppc64le)	IBM Z (s390x)
CentOS	✓		✓		
Debian	✓	✓	✓		
Fedora	✓				
Ubuntu	✓	✓	✓	✓	✓

Figure 7 Plateformes supportées Docker CE[8]

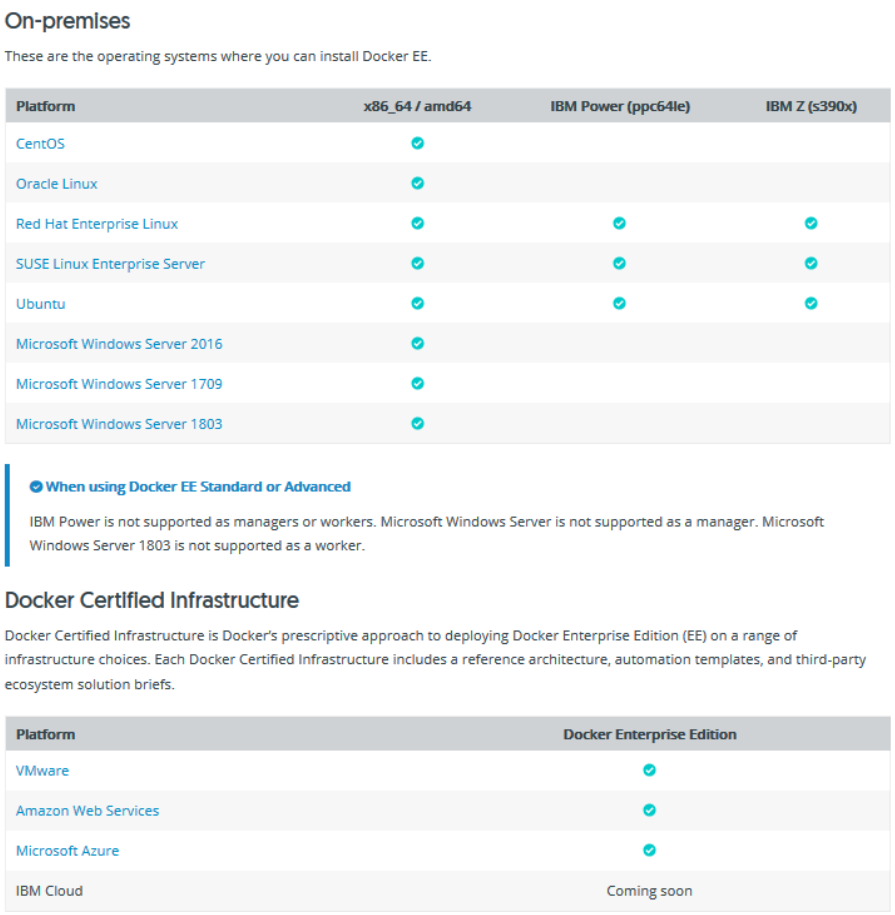
[7] https://docs.docker.com/install/#supported-platforms, consulté le 18 octobre 2018

[8] Capture d'écran de l'auteur depuis https://docs.docker.com/install/#supported-platforms, effectuée le 18 octobre 2018

2.2.2. Enterprise Edition

Le fonctionnement de base de la version entreprise de Docker (payante) est similaire à Docker CE mais elle offre des avantages, tels que le libre choix d'infrastructures, une meilleure portabilité des applications, une gestion unifiée, des scans de vulnérabilités dans les images, une possibilité d'intégration avec un annuaire LDAP (AD) d'entreprise, un support de Docker le jour même ou le lendemain etc. [9]

Cette version peut être installée sur des OS ou en solution Cloud, comme indiqué ci-dessous (Figure 7).



On-premises

These are the operating systems where you can install Docker EE.

Platform	x86_64 / amd64	IBM Power (ppc64le)	IBM Z (s390x)
CentOS	✓		
Oracle Linux	✓		
Red Hat Enterprise Linux	✓	✓	✓
SUSE Linux Enterprise Server	✓	✓	✓
Ubuntu	✓	✓	✓
Microsoft Windows Server 2016	✓		
Microsoft Windows Server 1709	✓		
Microsoft Windows Server 1803	✓		

When using Docker EE Standard or Advanced

IBM Power is not supported as managers or workers. Microsoft Windows Server is not supported as a manager. Microsoft Windows Server 1803 is not supported as a worker.

Docker Certified Infrastructure

Docker Certified Infrastructure is Docker's prescriptive approach to deploying Docker Enterprise Edition (EE) on a range of infrastructure choices. Each Docker Certified Infrastructure includes a reference architecture, automation templates, and third-party ecosystem solution briefs.

Platform	Docker Enterprise Edition
VMware	✓
Amazon Web Services	✓
Microsoft Azure	✓
IBM Cloud	Coming soon

Figure 8 Plateformes supportées Docker EE [10]

[9] Datasheet sur https://www.docker.com/products/docker-enterprise, consultée le 19.10.2018

[10] Capture d'écran de l'auteur depuis https://docs.docker.com/ee/supported-platforms/#supported-platforms, effectuée le 18 octobre 2018

2.3. Outils et méthodes Docker

2.3.1. Docker Swarm

Docker *Swarm* permet de distribuer les conteneurs entre plusieurs machines hôtes regroupées dans un *cluster* en fonction de certains facteurs, tels que la taille de mémoire nécessaire, l'espace de disque dur et la vitesse de *CPU*.

Jusqu'à la version 1.12 de Docker, *Swarm* fonctionnait en *standalone* (plus d'information sur https://docs.docker.com/swarm/). Depuis, il est remplacé par mode Docker *Swarm*.

Une fois le mode *Swarm* initialisé, deux rôles peuvent être définis :

- *Swarm manager* : la machine qui gère le fonctionnement du mode *Swarm* et tous les hôtes qui y sont connectés, appelés aussi *nodes* (nœuds en français). Il est recommandé d'utiliser un minimum de cinq *managers* en production afin de garantir une redondance optimale.
- *Swarm worker* : une machine hôte qui est connectée au mode *Swarm* et sur laquelle sont exécutés des conteneurs. Un nœud *manager* est par défaut également un *worker*.

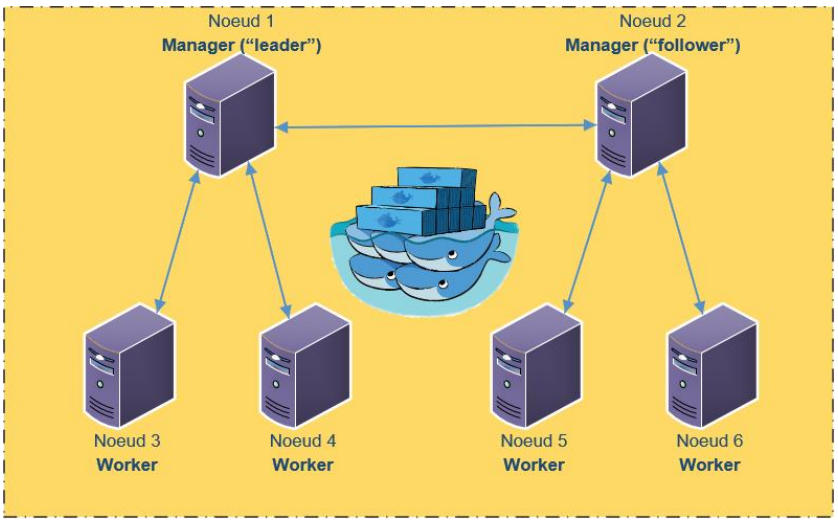


Figure 9 Swarm mode[11]

[11] Réalisée par l'auteur de ce document (2018)

2.3.2. Docker Compose

Une application peut être répartie sur plusieurs conteneurs. Docker Compose gère les services, les volumes et le réseau lors du lancement des conteneurs, à partir d'un fichier contenant des instructions YAML que l'on définit.

Docker Compose peut isoler les environnements les uns des autres via des noms de projet (par exemple éviter des conflits entre deux projets qui utilisent les mêmes noms de services, tels que « web », sur une machine hôte). De plus, des volumes utilisés lors de précédents conteneurs peuvent être récupérés dans le cas d'une nouvelle exécution. Enfin, en cas de non-modification d'un service donné, Docker Compose utilise les conteneurs créés auparavant.[12]

Vous pouvez voir un exemple de fichier *docker-compose.yml* ci-dessous (Figure 9). Plus d'informations sur https://docs.docker.com/compose/overview/.

docker-compose.yml

```yaml
version: "3"
services:
  #Un premier service appelé web
  web:
    # Spécification d'une image de base
    image: engel1605/get-started:part2
    deploy:
      # Exécuter 5 instances de l'image ci-dessus en tant que service appelé "web"
      replicas: 5
      # redémarrer les conteneurs en cas d'erreur
      restart_policy:
        condition: on-failure
      # Limiter l'utilisation du CPU à 10% et la mémoire RAM à 20 MB
      resources:
        limits:
          cpus: "0.1"
          memory: 20M
    # lier le port 4000 de l'hôte vers le port 80 (port web HTTP par défaut)
    ports:
      - "4000:80"
    # Définir le réseau que l'on veut utiliser pour ce service (webnet est un réseau avec load-balancing)
    networks:
      - webnet
  #2e service pour le visualisateur
  visualizer:
    image: dockersamples/visualizer:stable
    volumes:
      - "/var/run/docker.sock:/var/run/docker.sock"
    deploy:
      placement:
        #Faire en sorte que ce service soit déployé sur un noeud du swarm qui est manager
        constraints: [node.role == manager]
    ports:
      - "8080:8080"
    networks:
      - webnet
# Définition du réseau webnet avec réglages par défaut
networks:
  webnet:
```

Figure 10 Exemple de fichier Docker Compose

[12] https://docs.docker.com/compose/overview/#features, consulté le 23.10.2018

Remarque : l'indentation à l'intérieur du fichier docker-compose.yml est importante. De plus, les commentaires illustrés dans la figure 9 sont uniquement à titre indicatif dans ce rapport. Les commentaires, suivant où ils se situent dans le fichier, peuvent provoquer des erreurs d'exécution.

2.3.3. Docker Hub

Le registre Docker Hub est un service *Cloud* qui permet de mettre à disposition des autres utilisateurs une image que nous avons créée ou de télécharger des images officielles ainsi que celles de la communauté Docker.

Lorsque vous vous connectez (https://hub.docker.com), la page d'accueil vous montre les dépôts (*repository*) que vous avez déjà créés manuellement ou à travers le chargement de votre image sur Docker Hub. Vous pouvez aussi ajouter un nouveau dépôt.

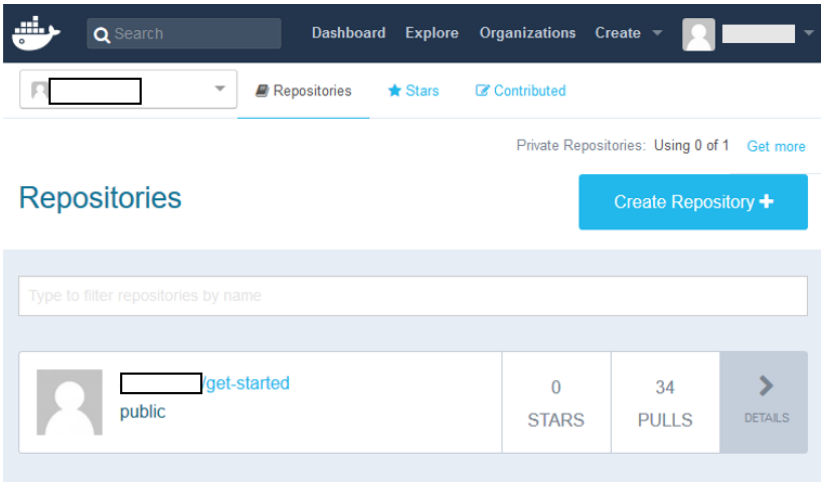


Figure 11 Page accueil Docker Hub[13]

[13] Capture écran réalisée par l'auteur depuis https://hub.docker.com/

Le bouton *Explore* situé dans la barre de navigation au sommet du site Docker Hub permet d'afficher une liste d'images Docker (dépôts) officielles comme par exemple *Ubuntu*, *Redis*.

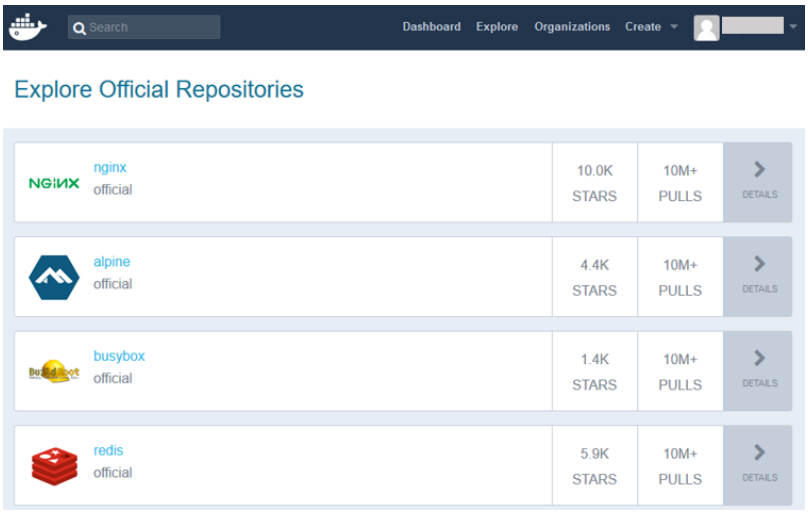


Figure 12 Exemple images officielles Docker Hub[14]

Il est également possible de créer une organisation sur Docker Hub. Une fois l'organisation créée, nous pouvons y ajouter des équipes et des membres et en modifier les paramètres comme illustré à la figure 12.

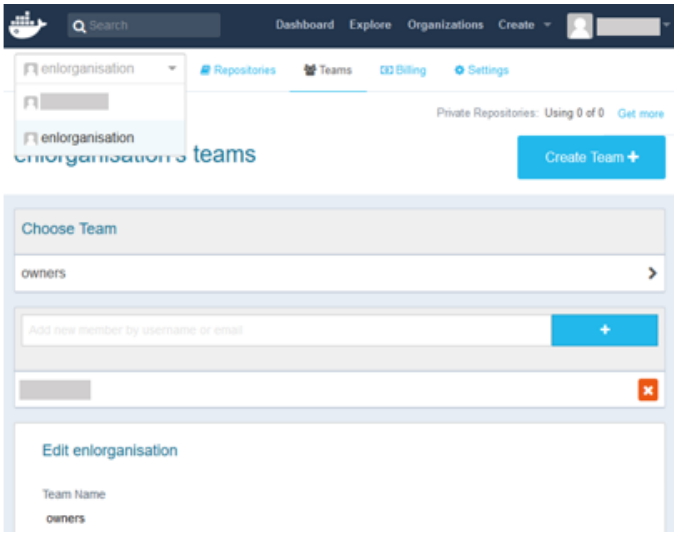


Figure 13 Affichage organisation sur Docker Hub[15]

[14] Capture écran réalisée par l'auteur depuis https://hub.docker.com/explore/

[15] Capture écran réalisée par l'auteur depuis https://hub.docker.com/u/enlorganisation/dashboard/teams/?team=owners

2.3.4. Docker Store

Docker Store sert de plateforme pour télécharger Docker CE et Docker EE, mais aussi des conteneurs officiels, d'autres images grâce à une option reliant le *Store* au Docker Hub et des modules Docker.

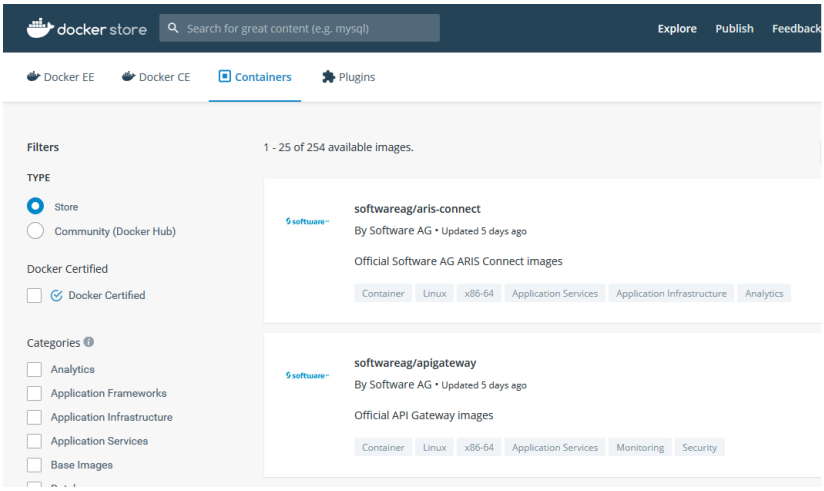


Figure 14 Docker Store[16]

2.3.5. Docker Cloud

Docker Cloud est un service payant qui permet de démarrer et gérer des machines Docker situées chez des fournisseurs Cloud comme Amazon Web Service ou DigitalOcean.

2.3.6. Docker Machine

Docker Machine est un outil qui permet d'installer et gérer Docker Engine sur plusieurs machines hôtes virtuelles localement ou dans un Cloud.

16 Capture écran réalisée par l'auteur depuis https://store.docker.com/search?q=&type=image&source=verified

2.3.7. Docker Bench Security app

Nous pouvons installer le script applicatif *Docker Bench Security* basé sur des *best practices* pour contrôler la configuration de la machine, du Docker daemon et de ses fichiers, les images et les conteneurs. Vous trouvez plus d'informations sur https://github.com/docker/docker-bench-security.

2.4. Standards / *best practices*

2.4.1. Taille des images Docker

Ce paragraphe traite de la taille des images, basé sur la documentation Docker en ligne. [17]

Pour commencer, les images devraient être aussi légères que possible afin de permettre des opérations plus rapides (réseau, mémoire et autres ressources). Pour cela, il faut d'abord sélectionner une image de base appropriée aux besoins (par exemple, éviter de prendre l'image *httpd* si un serveur Web Apache n'est pas nécessaire).

Ensuite, Docker 17.05 et les versions supérieures offrent la possibilité d'effectuer des *multi-stage builds*. Ceci signifie que nous pouvons écrire plusieurs instructions FROM dans le *Dockerfile*, ce qui a comme effet d'utiliser une image de base différente et de commencer une nouvelle étape de *build*. Ainsi, les artefacts créés dans une première étape peuvent par exemple être récupérés dans une deuxième, en éliminant les éléments inutiles. Cette fonction est utile par exemple pour compiler une application Java avec une image de base *java* pour après l'exécuter avec une image *Ubuntu*. Si l'utilisation du *multi-stage* est impossible, il est recommandé de combiner au maximum les instructions RUN.

[17] https://docs.docker.com/develop/dev-best-practices/#how-to-keep-your-images-small, consulté le 20.10.2018

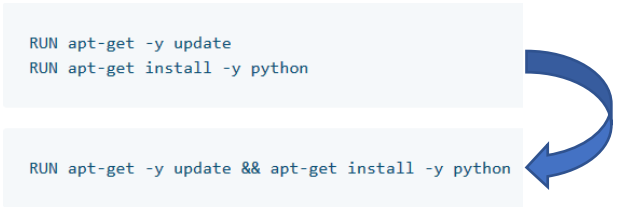


Figure 15 Combiner les instructions RUN[18]

En cas d'images avec des éléments similaires, il est préférable de créer notre propre base, dont les couches sont chargées en cache une seule fois, ce qui permet un chargement plus rapide et efficace.

Enfin, les étiquettes (*tags*) des images doivent être explicites (nom, version, canal de distribution), que ce soit dans le *Dockerfile* avec les instructions LABEL ou MAINTAINER ou lors du *build* de l'image (avec l'option *-t <nom du tag>*).

2.4.2. Persistance des données

Docker conseille d'utiliser les volumes, *bind mounts* et *secrets* pour stocker des données d'applications, à la place des pilotes qui permettent le stockage dans la couche « écriture » du conteneur.[19]

2.4.3. Fichier .dockerignore

Nous pouvons écrire un fichier *.dockerignore* qui contient une liste de répertoires / fichiers que le processus Docker doit ignorer lors de la création d'un conteneur. Cela permet d'éviter l'utilisation de contenus larges ou sensibles.[20]

---

[18] Capture d'écran de l'auteur depuis https://docs.docker.com/develop/dev-best-practices/#how-to-keep-your-images-small, réalisée le 20.10.2018

[19] https://docs.docker.com/develop/dev-best-practices/#where-and-how-to-persist-application-data, consulté le 20.10.2018

[20] https://docs.docker.com/engine/reference/builder/#dockerignore-file, consulté le 24.10.2018

2.4.4. Sécurité

Comme indiqué par R. Mckendrick *et al.* (2017), il existe certaines *best practices* de base en ce qui concerne la sécurité.

Pour commencer, il est fortement recommandé d'intégrer une seule application (ou module) par conteneur et d'installer le strict nécessaire au bon fonctionnement, ceci permettant de réduire les zones de cyber-attaque.

Ensuite, il faut accorder une attention particulière aux droits d'accès (qui, où, quand, comment, pourquoi) sur les images (et conteneurs).

Troisièmement, comme tout logiciel ou matériel, il faut maintenir un suivi des versions et des *patchs* de sécurité apportés par Docker et effectuer ces mises-à-jour si possible en fonction de l'utilisation en entreprise.

Enfin, de nombreuses documentations, des forums et les plateformes Docker sont mis à disposition et maintenus à jour par la communauté.

3. Analyse des risques

Ce chapitre est consacré à l'analyse des risques liés à l'exécution d'images Docker qui contiennent du code étranger arbitraire écrit par les chercheurs sur une infrastructure interne avec un accès notamment à des données sensibles. Ces risques sont établis à partir de faits répertoriés ou en fonction de nos analyses et tests effectués dans ce travail. Nous étudions la question sous trois angles : les dommages sur l'infrastructure, les risques de sécurité et les implications légales.

3.1. Infrastructure

Les risques mentionnés ci-dessus sont liés à Docker et ils peuvent se retrouver dans notre cas d'utilisation.

3.1.1. Surcharge de ressources

Un premier risque est une détérioration matérielle ou une extinction non-désirable d'une ou plusieurs machines liée à une utilisation excessive de la mémoire vive (RAM), une surchauffe des processeurs (CPU) ou une surcharge au niveau des cartes graphiques.

Pour éviter cela, nous analysons la possibilité de fixer des quotas de RAM et CPU plus loin dans ce document. De plus, Docker *Swarm* utilise une ou l'autre des stratégies, ainsi que des filtres qui peuvent être définis dans un nouveau service, pour permettre de distribuer et démarrer un conteneur sur le meilleur nœud disponible. Par exemple, nous pouvons ajouter une étiquette « HighRAM == true » à un nœud qui offre une haute capacité mémoire et les tâches dans un service Docker qui contient cette contrainte sont distribuées sur cette machine.

3.1.2. Utilisation du stockage

L'espace de stockage des disques durs peut être rempli par des conteneurs Docker qui n'ont plus lieu d'être ou qui contiennent des images trop larges par rapport à l'utilisation qui s'y rapporte.

Nous préconisons donc un moyen (automatique ou non) de supprimer à intervalles réguliers ces conteneurs.

Il est possible d'établir une limite de taille avec l'option *--storage-opt size=<taille souhaitée>* (ex : *docker run --storage-opt size=50G ubuntu*). Cependant, comme nous constatons lorsque

nous exécutons cette commande sur le serveur, cette option ne peut être appliquée qu'avec *overlay* si le système de fichier *backup* est en xfs et que l'option de montage *pquota* est utilisée. Ceci n'est pas le cas pour notre installation de Docker puisque nous avons un système de fichier *backup* en extfs.

3.1.3. Panne / disfonctionnement

Troisièmement, nul n'est à l'abri d'une panne ou disfonctionnement. Par exemple, un nœud *manager* du *Swarm* peut être soudainement inaccessible et cela a une conséquence sur la création de services Docker et la répartition des tâches. Il est donc vivement conseillé d'assurer une redondance entre les machines et les équipements réseau.

3.2. Sécurité

Docker permet d'exécuter des programmes et codes dans un environnement *sandbox* (conteneurs). Mais, comme toute application de type *Cloud*, cela peut présenter des risques sécuritaires qui ne doivent pas être négligés, d'autant plus que des données médicales (donc sensibles) seront analysées et que les images Docker accéderont à un réseau interne. Nous allons décrire les menaces les plus plausibles et proposer pour chacune des actions de prévention.

3.2.1. Menaces / failles possibles

Les points 3.2.1.1 à 3.2.1.7 de ce chapitre sont basés sur une publication de la société Sysdig[21], ainsi que sur le rapport de Robail Y. (2018).

3.2.1.1. Kernel et hôte Docker

Il est déconseillé d'utiliser des conteneurs Docker sur des machines vulnérables et compromises. Celles-ci doivent donc être sécurisées par l'emploi de mots de passes appropriés, des contrôles d'accès et des communications cryptées. De plus, il est important d'utiliser un système d'exploitation « minimal » approprié aux besoins et celui-ci doit être mis-à-jour régulièrement, tout comme l'instance Docker qui y est installée.

[21] https://sysdig.com/blog/7-docker-security-vulnerabilities/, consulté le 20.11.2018

Enfin, le processus Docker utilise le module de sécurité Linux *AppArmor* qui permet de créer et appliquer des profils sur les applications (et conteneurs Docker). Plus d'informations sur https://docs.docker.com/engine/security/apparmor/.

3.2.1.2. Authenticité des images

Les images Docker peuvent être accédées et téléchargées depuis Docker Hub ou d'autres dépôts en ligne. Il est donc nécessaire de vérifier la provenance de ces images (de préférence avec des preuves de sécurité à l'appui) et le développeur de l'image doit être considéré comme fiable.

Il est recommandé d'activer *DOCKER_CONTENT_TRUST* (*export DOCKER_CONTENT_TRUST=1*) pour autoriser uniquement le téléchargement d'images de confiance (« signées »).

Il est possible d'installer un serveur de registre (dépôt) Docker interne à l'entreprise, pour que les images restent « sous notre contrôle », ce qui est un objectif secondaire du cahier des charges, qui ne sera pas implémenté par manque de temps restant.

3.2.1.3. Container breakout

Ce terme *container breakout* représente le fait qu'un conteneur a réussi à passer outre les vérifications d'isolation et peut accéder à des informations sur la machine hôte interne ou acquérir des accès privilégiés.

Comme indiqué dans les *Best practices Docker*, il faut éviter dans la mesure du possible de démarrer des conteneurs avec des accès « super-utilisateur » (*root* par exemple).

Par ailleurs, comme indiqué dans la documentation Docker, les conteneurs ont la possibilité d'être isolés à travers des *user namespaces*.[22]

3.2.1.4. Surcharge des ressources (déni de service)

Le principe d'une attaque de type « déni de service » (*Denial-of-Service* en anglais ou DoS) est de surcharger l'utilisation des ressources telles que la mémoire, les processeurs et les

[22] https://docs.docker.com/engine/security/userns-remap/, consulté le 23.11.2018

interfaces réseau pour rendre un service inutilisable ou pour provoquer un manque de ces ressources pour d'autres processus (et conteneurs dans notre cas). Tel qu'indiqué précédemment, nous établissons des limites de mémoire et de processeur au niveau du service Docker.

3.2.1.5. Man-in-the-Middle

Le terme *Man-in-the-Middle* (« Homme au milieu » en français) fait référence à l'intrusion une personne malintentionnée (*hacker*) au sein d'un réseau entre deux partis (par exemple entre un service Web et un autre serveur) pour visualiser, modifier, supprimer ou voler des informations transmises lors d'une communication.

Une solution à ce problème est d'isoler les réseaux, tel que nous le faisons plus loin dans ce document, afin d'éviter une intrusion externe dans le conteneur. De plus, nous pouvons utiliser des réseaux privés virtuels (*Virtual Private Network* ou VPN en anglais), mais nous ne nous attardons pas sur cette option à cause de manque de temps et ce n'est pas un objectif du cahier des charges.

Pour finir, il faut utiliser le protocole Web sécurisé (HTTPs) au lieu de HTTP, d'autant plus que dans notre cas, des résultats de recherche transitent entre un serveur et l'interface Web cliente notamment.

3.2.1.6. ARP Spoofing

Les réseaux utilisent ce qu'on appelle des tables ARP (*Address Resolution Protocol*), ce qui permet de lier l'adresse physique unique (MAC) d'une machine à une adresse IP. Cette attaque permet donc à un *hacker* de lier son adresse MAC à une adresse IP du réseau interne (donc légitime) afin de surveiller, sniffer le trafic réseau et les informations sensibles qui y transitent, voire injecter des virus ou maliciels (*malwares* en anglais). Docker offre la possibilité d'utiliser *ebtables* pour filtrer le trafic et bloquer ces usurpations.

3.2.1.7. Crédenciales et Docker secrets

Tous les mots de passes et les clés de sécurité doivent être protégés de façon adéquate. Par ailleurs, il est recommandé de ne pas utiliser des variables d'environnement pour accéder à des *Docker secrets* et de régler les conteneurs en lecture seule.

3.2.1.8. Modification de données

Le code des chercheurs accède à des données sensibles. Il est donc absolument primordial que les données et leurs emplacements ne puissent pas être modifiés ou supprimés.

Lorsque nous ajoutons un point de montage avec *bind mount* dans un conteneur, il faut que celui-ci soit en lecture seule (*readonly*).

3.2.1.9. Accès à des volumes de données non-autorisés

Il est important que le code d'un chercheur accède uniquement aux données correspondantes aux besoins de celui-ci, pour éviter des erreurs d'analyse ou un accès non-autorisé.

Le service Web doit donc créer les points de montage vers les volumes corrects dans le conteneur Docker.

3.2.1.10. Fuite de données vers l'extérieur

Les informations sensibles sont une mine d'or pour les *hackers*. Il est donc nécessaire, en plus des éléments cités précédemment, d'isoler le réseau d'un conteneur pour éviter une communication de ces données vers l'extérieur, ce qui peut avoir de graves conséquences au niveau des données du patient, pour la réputation de l'institution impactée et par rapport à la loi.

3.2.1.11. Accès non-autorisé aux résultats d'analyse

Comme expliqué dans ce rapport, les chercheurs du milieu hospitalier demandent l'exécution d'un conteneur avec l'image Docker souhaitée et les résultats sont affichés sur l'interface Web. Il est donc recommandé de mettre en place un mécanisme d'authentification sur cette interface (lien avec la session de l'ordinateur professionnel, ajout d'une page de *login*) pour restreindre l'accès à ces résultats uniquement aux chercheurs concernés par la recherche.

3.2.2. Solutions d'analyse / protection

La société Sysdig entre autres a établi une liste de 29 outils de sécurité (gratuits et payants). [23]Nous en décrivons trois dans ce chapitre.

3.2.2.1. Docker bench security

Docker offre un script intéressant qui analyse la configuration sur une machine hôte en fonction de *best practices* : *Docker-bench-security* (Figure 15). Pour le lancement du script et des étapes pour corriger les avertissements, vous pouvez vous rendre sur https://www.digitalocean.com/community/tutorials/how-to-audit-docker-host-security-with-docker-bench-for-security-on-ubuntu-16-04.

Nous nous basons sur des points de ce script pour mettre en place notre infrastructure Docker locale.

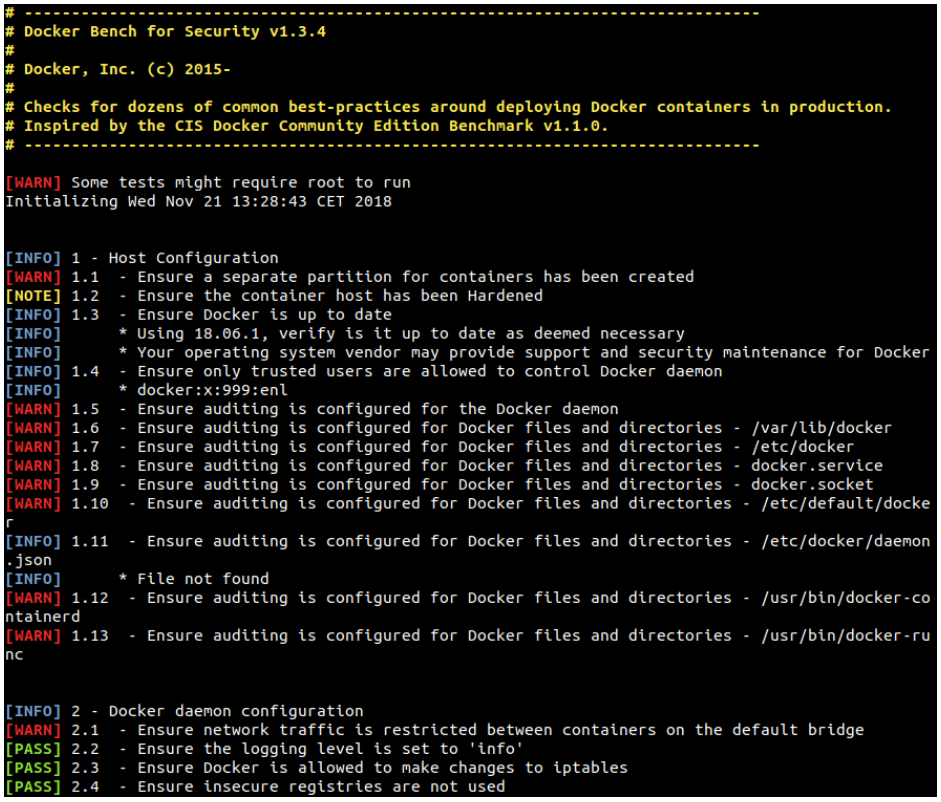


Figure 16 Début du script Docker-bench-security[24]

[23] https://sysdig.com/blog/20-docker-security-tools/, consulté le 27.11.2018

[24] Capture d'écran réalisée par l'auteur de ce document à partir du serveur de tests utilisé

3.2.2.2. Sysdig Secure

La compagnie Sysdig propose son produit *Sysdig Secure* qui permet de réaliser des audits de sécurité, une gestion des vulnérabilités, des scans d'images Docker, des analyses de performances et de conformité, des opérations forensiques (par exemple : reproduction d'intrusions, fuite de données). Vous trouvez plus d'informations sur https://sysdig.com/products/secure/.

Vous pouvez leur demander une version Cloud ou logicielle (installation sur un serveur), les prix variant en fonction du contexte d'utilisation. D'après leur site, il faut souscrire pour un minimum d'une année. Nous ne testons donc pas ce logiciel dans ce travail.

3.2.2.3. Sysdig et Sysdig Inspect

Le logiciel open-source gratuit *Sysdig Inspect* rend possible l'analyse des captures de performances, du trafic et de la sécurité de conteneurs Docker effectuées au préalable par l'outil *Sysdig*, qui peut être installé sur une machine hôte ou exécuté dans un conteneur. Plus d'informations sur :

- https://sysdig.com/opensource/inspect/
- https://github.com/draios/sysdig/wiki/How-to-Install-Sysdig-for-Linux (installation de *Sysdig*)
- https://github.com/draios/sysdig-inspect/

Ce logiciel ne permet à priori pas d'effectuer un contrôle d'image Docker « en temps réel », avant le lancement de celle-ci. De plus, nous sommes obligés de posséder des fichiers de captures générés par *sysdig*, donc les analyses sont faites par après. Ceci est utile tout-de-même pour effectuer des analyses forensiques sur les conteneurs qui sont exécutés sur un hôte Docker et résoudre des problèmes éventuels.

3.2.2.4. Anchore Engine

L'outil Anchore Engine est un outil personnalisable pour analyser des images Docker, leur contenu et leurs potentielles vulnérabilités, à partir de règles (*policies*) par défaut ou définies par l'utilisateur. Ces images sont contrôlées dans un registre ou avant qu'elles soient déployées dans des conteneurs. Nous ne testons pas en détail cet outil, par manque de temps, mais il peut se révéler pratique pour posséder des images saines. Pour installer l'outil, veuillez vous référer à https://github.com/anchore/anchore-engine. La documentation se trouve sur https://anchore.freshdesk.com/support/home.

3.3. Implications légales

Le traitement de données personnelles (et sensibles) est, de nos jours, un sujet très important et récurrent et il est soumis aux lois de protection des données.

3.3.1. Principes de base de protection des données

Comme indiqué par exemple par le Centre National de Recherche Scientifique (CNRS), il existe sept principes de base qui doivent être pris en compte lors d'un traitement des données :

- Finalité
- Proportionnalité
- Pertinence des données
- Conservation
- Sécurité / confidentialité
- Transparence
- Respect du droit des personnes

3.3.1.1. Finalité

Les données recueillies doivent être traitées uniquement pour satisfaire les objectifs légitimes déterminés au moment de la collecte. (Engel, 2018)

3.3.1.2. Proportionnalité et pertinence des données

Les données doivent être « *nécessaires pour leur finalité […] adéquates, pertinentes et non excessives au regard des objectifs poursuivis.* » (CNRS, 2018)[25]

3.3.1.3. Conservation des données

Il est possible de distinguer deux périodes : la conservation, durant laquelle les données sont accessibles en consultation et l'archivage des données, durant laquelle les données sont stockées ailleurs dans l'entreprise et ne sont en principe plus utilisées. Les durées définies peuvent varier selon les lois. (Engel, 2018)

[25] http://www.cil.cnrs.fr/CIL/spip.php?article3055, consulté le 02.07.2018

3.3.1.4. Sécurité et confidentialité

Les données doivent être sécurisées en fonction de leurs types et des risques, en mettant en place des accès réservés aux utilisateurs et services autorisés (internes ou externes à l'entreprise), une protection physique (sauvegardes, contre les dégâts d'eau et feu et autres), une couverture anti-virus et contre d'autres attaques (mots de passe complexes, cryptage). (Engel, 2018)

3.3.1.5. Transparence

Une personne dont les données sont récoltées, traitées ou transmises à des tiers doit en être avertie de façon explicite. (Engel, 2018)

3.3.1.6. Respect du droit des personnes

Les citoyens possèdent des droits relatifs à l'utilisation de leurs données, tels que le droit à l'information, le droit d'accès, le droit de rectification et le droit d'opposition au traitement de leurs données. (Engel, 2018)

3.3.2. Lois concernées

Dans le cadre de la recherche sur des données médicales en Suisse, les quatre lois suivantes doivent être prises en compte:

- la loi fédérale suisse relative à la recherche sur l'être humain (LRH)
- la loi cantonale concernée pour la protection des données
- la loi fédérale suisse pour la protection des données (LPD)
    - Cette loi s'applique aux traitements effectués par un organe fédéral ou pour le compte de celui-ci
    - *« Les traitements de données effectués par des organes publics cantonaux ou communaux, tels les universités (à l'exception des EPF) et les hôpitaux cantonaux et universitaires, régionaux et communaux, ressortent de la compétence des autorités cantonales ou communales de protection des données »*(Préposé fédéral à la protection des données et à la transparence, 2018)[26]. La LPD n'est donc pas concernée, ou du moins pas en priorité.

[26] https://www.edoeb.admin.ch/edoeb/fr/home/protection-des-donnees/statistik--register-und-forschung/recherche/protection-des-donnees-et-recherche-en-general.html, consulté le

    - Cette loi est en cours de révision par le gouvernement suisse (P-LPD)
- le règlement européen sur la protection des données (RGPD)
    - Le RGPD s'applique dans notre cas si la recherche concerne des patients résidant dans l'Union européenne et le suivi de leurs comportements sur ce territoire. Le contexte d'application pratique de cette loi est donc encore relativement « abstrait ».

3.3.3. Anonymisation et pseudonymisation

La CNIL résume le principe d'anonymisation et pseudonymisation, repris ci-dessous. [27]

Le procédé d'anonymisation de données personnelles vise à supprimer toute donnée permettant d'identifier directement un individu (nom, prénom, date de naissance, adresse, numéro AVS, etc.). En principe, les lois mentionnées plus haut ne s'applique pas aux données qui ont été **totalement anonymisées**. Cependant, comme indiqué par la CNIL, il est difficile à effectuer, car avec les quantités de données actuelles, une personne pourrait être identifiée par ses habitudes de vie et ses comportements.

Il est donc possible de réaliser une pseudonymisation, c'est-à-dire que les données directement liées à une personne sont séparées du reste et protégées avec une clé d'identification. Il faut toutefois être attentif à la protection optimale de ces données isolées et de leur clé.

3.3.4. Conditions d'utilisation des données

Le patient a le droit d'être informé de manière claire et précise de l'utilisation de ses données personnelles, avec des indications comme le but du traitement, les acteurs et la durée et les chercheurs doivent être conscients et respecter les normes et conditions mises en place.

C'est pourquoi il faut mettre en place un mécanisme de consentement affirmatif avec signature pour le patient et un contrat d'utilisation des données pour les chercheurs. Ceci est un objectif du cahier des charges de ce travail.

23.11.2018

[27] http://www.cil.cnrs.fr/CIL/spip.php?article3056, consulté le 19.11.2018

3.3.5. Fuite ou modification non-autorisée de données

En ce qui concerne la notification auprès des autorités en cas de faille sur les données personnelles, *« L'art. 22 al. 1 nLPD prévoit que le responsable du traitement notifie sans délai au Préposé fédéral à la protection des données et à la transparence suisse (PFPDT) tout traitement non autorisé ou toute perte de données personnelles. L'art. 33 RGPD prévoit une obligation similaire. Dans le cas d'un responsable du traitement établi en Suisse mais soumis au RGPD [18], l'annonce exigée par le RGPD devrait être faite non pas au PFPDT mais à l'autorité de contrôle compétente de chaque Etat membre où des personnes sont concernées par la faille [19]. Ce n'est que si la violation ne présente vraisemblablement pas de risques pour la personnalité et les droits fondamentaux des personnes concernées que le responsable du traitement peut y renoncer.»* (Me Sylvain Métille, 2018)[28]

3.3.6. Sanctions en cas de non-respect des lois

Le RGPD prévoit des sanctions allant d'un avertissement, une mise en demeure et des amendes pouvant monter jusqu'à 20 millions d'Euros ou 4% du chiffre d'affaire. La LPD, en revanche, ne prévoit pas de sanctions spécifiques, mise à part une amende pouvant aller jusqu'à CHF 10'000.- ou une procédure judiciaire en cas de non-respect par rapport à certaines décisions devant être prises par le préposé fédéral à la protection des données suisse.

[28] https://www.hdclegal.ch/wp-content/uploads/2017/11/METILLE_Faille-et-securite_Expert-focus_11-2017.pdf, consulté le 24.11.2018

4. Choix effectués

4.1. Méthodologie / planification

Pour la réalisation de ce projet, nous nous sommes basés sur la méthodologie Agile SCRUM. Etant donné que ceci est un travail individuel, nous ne pouvons pas utiliser cette méthodologie telle quelle et nous devons l'adapter.

Pour commencer, un cahier des charges (cf. Annexe I) est établi entre le professeur qui suit ce travail de Bachelor, le répondant technique (M. Ivan Eggel) et l'auteur de ce document, dans lequel les objectifs primaires (*must have*) et secondaires (*nice to have*) sont établis.

A partir de ce cahier des charges signé, un fichier Excel servant de *Product Backlog* est rédigé. Celui-ci contient les fonctionnalités / composantes du projet à réaliser sous forme d'*User stories* (selon la méthodologie Agile) et permet à l'étudiant de se repérer dans l'avancement du travail.

Chaque semaine, une rencontre a lieu entre le professeur, le répondant technique et l'étudiant afin d'assurer un suivi dans le déroulement du travail.

4.2. Technique (matériel)

Un serveur virtuel avec comme système d'exploitation Linux Ubuntu 18.04 LTS est fourni par le Service informatique de la HES-SO Valais. Il contient une installation de Docker et le service Web et le site servant d'interface cliente (démonstration). Il permet également la réalisation des tests liés à l'analyse de Docker et aux développements.

Deux machines virtuelles locales Linux (sur l'ordinateur de développement) sont par ailleurs utilisées :

- La première afin d'ajouter un nœud dans le Docker *Swarm* local mis en place sur le serveur ;
- La deuxième pour le développement de l'interface Web cliente.

4.3. Technique (logiciel)

Les choix logiciels sont basés sur les connaissances déjà acquises, ainsi que les développements réalisés au préalable.

En ce qui concerne le service Web permettant la communication entre les interfaces clientes et l'API Docker, le choix se porte sur une application de type Spring Boot avec le protocole REST relativement facile à utiliser. L'application est réalisée dans le logiciel Eclipse (version *Oxygen*)[29], puis déployée sur le serveur.

L'interface Web cliente est réalisée en NodeJS[30] à l'aide du programme WebStorm[31], langage relativement récent et évolutif.

[29] https://www.eclipse.org/oxygen/

[30] https://nodejs.org/en/

[31] https://www.jetbrains.com/webstorm/

5. Résultats

5.1. Architecture générale

Le schéma ci-dessous illustre l'architecture mise en place dans ce projet.

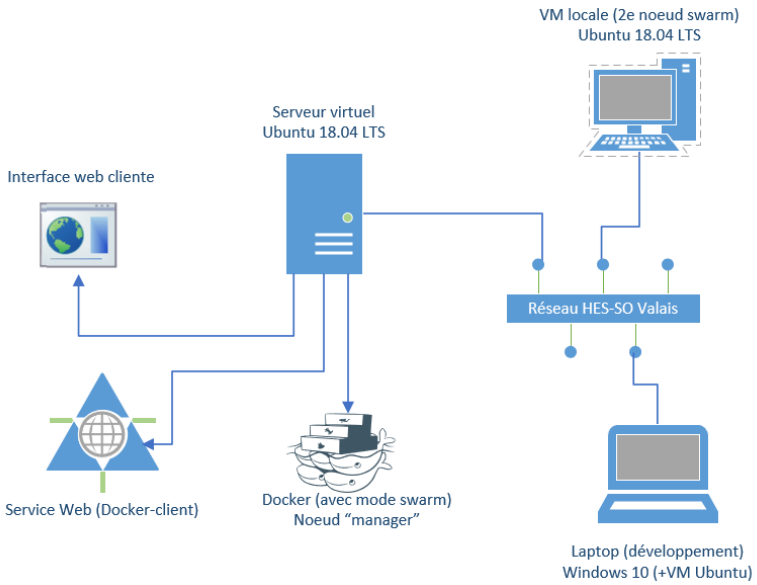


Figure 17 Architecture générale de ce projet[32]*. Un serveur Ubuntu est configuré et connecté au réseau informatique de la HES-SO Valais. Sur ce serveur, nous installons Docker avec le mode* Swarm *(il a ainsi le rôle de* manager*), nous mettons en place le service web qui utilise le client spotify/docker et nous configurons un conteneur Docker qui exécute notre interface web cliente. Comme indiqué dans les choix techniques de matériel, nous avons sur ce réseau une machine virtuelle qui joue le rôle de nœud* worker *dans notre* Swarm *local. Enfin, nous rédigeons le rapport et nous effectuons les développements sur un ordinateur portable également connecté sur le réseau.*

5.2. Installation Docker et mode *Swarm*

La réalisation pratique dans ce projet vise avant tout à installer Docker sur le serveur Ubuntu fourni et mettre en place le mode *Swarm*.

Comme indiqué précédemment, nous installons la version *Community* de Docker. Pour cela, nous suivons les instructions d'installation à partir d'un dépôt : *Install using the repository*[33].

[32] Diagramme réalisé à l'aide de Microsoft Visio par l'auteur de ce document

[33] Consulté le 8 octobre 2018

Par la suite, nous initialisons le mode *Swarm* via la commande *sudo docker swarm init*. Cela a pour effet de créer ce mode et d'y ajouter le serveur en tant que nœud *manager*. Nous ajoutons ensuite un nœud *worker* sur lequel Docker est préalablement installé.

Nous pouvons voir ci-dessous que Docker est bien installé (Figure 17). Le mode *Swarm* est initialisé et il contient deux nœuds (Figure 18).

```
[redacted]:~$ sudo docker swarm join-token worker
To add a worker to this swarm, run the following command:

    docker swarm join --token SWMTKN-1-2xyjii9crp8j7xvtbgwoncx9msu29z453andvt5fc
devk9shwo-78iyvboxsa5v7o8u9ug0gpn2r [redacted] :2377
```

Adresse manager

Figure 18 Commande pour ajout nœud Swarm

```
[redacted]:~$ sudo docker info
Containers: 2
 Running: 2
 Paused: 0
 Stopped: 0
Images: 17
Server Version: 18.06.1-ce
Storage Driver: overlay2
 Backing Filesystem: extfs
 Supports d_type: true
 Native Overlay Diff: true
Logging Driver: json-file
Cgroup Driver: cgroupfs
Plugins:
 Volume: local
 Network: bridge host macvlan null overlay
 Log: awslogs fluentd gcplogs gelf journald json-file logentries splunk syslog
Swarm: active
 NodeID: u2fv8nr6aq5bzfybmmnvjeher
 Is Manager: true
 ClusterID: zgn1xr0nv7catda2rrzs72krx
 Managers: 1
 Nodes: 2
```

Figure 19 Affichage infos Docker

```
[redacted]$ sudo docker node ls
ID                            HOSTNAME        STATUS  AVAILABILITY  MANAGER STATUS  ENGINE VERSION
u2fv8nr6aq5bzfybmmnvjeher *   [redacted]      Ready   Active        Leader          18.06.1-ce
icqz0pb5ohfyf04nrfweariy8     vmtestdocker    Down    Active                        18.06.1-ce
```

Figure 20 Affichage des nœuds du Swarm

Comme indiqué sur https://www.digitalocean.com/community/tutorials/how-to-audit-docker-host-security-with-docker-bench-for-security-on-ubuntu-16-04, nous avons configuré le *daemon* Docker pour que la communication entre conteneurs sur un réseau *bridge* (cf. point 5.6) soit restreinte par défaut, qu'un pilote de logs soit défini et qu'il ne puisse pas y avoir une augmentation des privilèges utilisateurs à l'intérieur d'un conteneur.

De plus, nous installons le trousseau de clés *pass* qui permet de stocker les mots de passe de façon plus sécurisée lors de la connexion sur Docker Hub (*docker login*).[34] Remarque : si la génération de clé GPG s'exécute dans le vide, installez *rng-tools* et exécutez la commande *sudo rngd -r /dev/urandom*.[35]

5.3. Exécution de conteneurs (Service web)

Nous avons vu que des conteneurs peuvent être créés et démarrés en utilisant le client Docker en ligne de commande.

Or, un objectif du cahier des charges est d'utiliser le client Docker Java Spotify/docker-client dans un service Web pour permettre la communication avec l'API de l'instance Docker installée sur le serveur.

Comme nous l'avons dit, nous nous basons sur un modèle de Spring Boot Application, généré depuis l'adresse https://start.spring.io/, que nous développons par la suite via Eclipse.

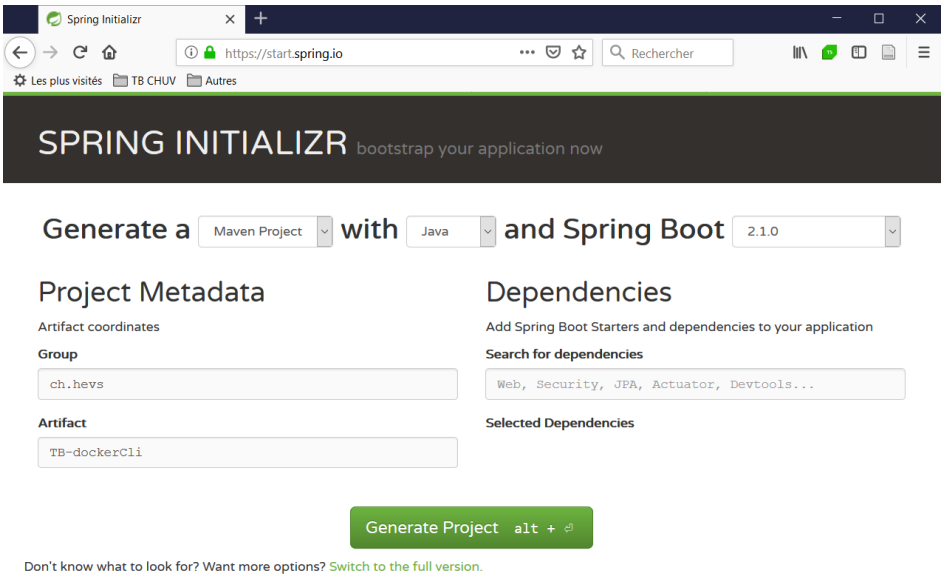


Figure 21 Génération d'un modèle Spring Boot App[36]

[34] https://github.com/docker/docker-credential-helpers/issues/102, consulté le 26.11.2018

[35] https://delightlylinux.wordpress.com/2015/07/01/is-gpg-hanging-when-generating-a-key/, consulté le 26.11.2018

[36] Capture d'écran à partir de https://start.spring.io/, réalisée le 8 novembre 2018

5.3.1. Création de base d'un service Docker

Dans ce point, nous décrivons pas-à-pas la fonction que nous avons écrite dans le service Web. Par la suite, nous ajouterons des éléments à cette fonction (spécificités du conteneur, quotas RAM et CPU, définition du réseau, conditions de redémarrage) .

1) Nous définissons la fonction, ainsi que les variables qui seront passées en paramètre lors de l'appel de cette fonction. Dans notre cas, nous avons trois variables : le nom du service qui doit être créé (*serviceName*), l'identifiant de l'image Docker utilisée (*imageId*) et un type de nœud du *Swarm* sur lequel nous voulons que le conteneur soit exécuté (*nodeLabel*).

```
@PostMapping(value="/services/create", params={"serviceName", "imageId", "nodeLabel"})
    public String createService(
            @RequestParam("serviceName") String serviceName,
            @RequestParam("imageId") String imageId,
            @RequestParam("nodeLabel") String nodeLabel)
            throws DockerException, InterruptedException {

    }
```

Figure 22 Création de la fonction serviceCreate (dans service Web)

2) Nous indiquons au client Docker du serveur (sur lequel notre service Web est déployé) qu'il doit télécharger l'image Docker souhaitée à partir du Hub. Nous définissons ensuite une variable *containerSpecs* qui contient les spécificités du conteneur créé, comme dans notre cas l' identifiant de cette image, ainsi que les *bind mount* vers les points de montage souhaités.

```
//Récupérer l'image souhaitée depuis le dépôt Docker
docker.pull(imageId);

//Ajout des bind-mount vers les points de montage NAS
//Source: chemin de la machine hôte sur lequel nous voulons pointer
//Target: chemin dans le conteneur sur lequel nous relions la source
final Mount bind1 = Mount.builder()
        .type("bind")
        .source("                ")
        .target("       ")
        .readOnly(true)
        .build();
final Mount bind2 = Mount.builder()
        .type("bind")
        .source("                ")
        .target("       ")
        .readOnly(true)
        .build();
final List<Mount> binds = new ArrayList<>();
binds.add(bind1);
binds.add(bind2);

//Définition de l'image Docker que le conteneur va exécuter
final ContainerSpec containerSpecs = ContainerSpec.builder()
        .image(imageId)
        .mounts(binds)
        .build();
```

Figure 23 Spécificités liées au conteneur (dans service Web)

3) Nous indiquons les spécificités de base liées à la tâche qui sera déployée par le service Docker créé :
   a) la contrainte de placement *placeConstraints* avec *placeSpecs* (filtre que le *Swarm* utilisera pour déployer sur un type de nœud souhaité) à partir du paramètre *nodeLabel* spécifique
   b) les conditions de redémarrage de la tâche (en cas d'échec et jusqu'à cinq tentatives au maximum)
   c) la variable *taskSpecs* reprend les spécificités du conteneur et les éléments de a) et b)

```
        //Paramètre qui indique au service sur quel noeud du Swarm
        //doit être envoyé le conteneur
        final List<String> placeConstraints = new ArrayList<>();
        placeConstraints.add("node.labels." + nodeLabel + "==true");

        final Placement placeSpecs = Placement.create(placeConstraints);


        //Conditions de redémarrage de la tâche dans le service
        //(ici, redémarrer en cas d'échec, jusqu'à 5 tentatives max)
        final RestartPolicy restartPolicy = RestartPolicy.builder()
                .condition(RestartPolicy.RESTART_POLICY_ON_FAILURE)
                .maxAttempts(5)
                .build();

        //Définition de la tâche à démarrer dans le service
        final TaskSpec taskSpecs = TaskSpec.builder()
                .containerSpec(containerSpecs)
                .restartPolicy(restartPolicy)
                .placement(placeSpecs)
                .build();
```

Figure 24 Spécificités liées à la tâche (dans service Web)

4) Nous indiquons les spécificités du service avec : le nom, les spécificités de la tâche définie ci-dessus et le mode de service (ici, répliqué une fois). Puis nous envoyons une requête de lancement du service sur le client Docker. La fonction nous renvoie en retour l'identifiant du nouveau service.

```
        //Définition du service avec son nom, la tâche, le mode
        //(répliqué 1 fois => 1 seul conteneur) et le réseau
        final ServiceSpec serviceSpecs = ServiceSpec.builder()
                .name(serviceName)
                .taskTemplate(taskSpecs)
                .mode(ServiceMode.withReplicas(1L))
                .build();


        //Création du service dans l'instance Docker client
        //créée ci-dessous (dans la méthode main)
        final ServiceCreateResponse newService = docker.createService(serviceSpecs);

        return "Service " + newService.id() + " created!";
    }
```

Figure 25 Spécificités et lancement du service Docker (dans service Web)

5.4. Quotas RAM

Pour commencer, si un conteneur nécessite plus de mémoire RAM dans son fonctionnement que le système d'exploitation d'une machine ne le permet, ceci peut conduire à un arrêt non-désirable d'autres applications et de processus (par ex : Docker) commandé par le système pour libérer de l'espace ou dans le pire des cas un arrêt complet. Nous pouvons éviter ça en définissant un taux maximum de mémoire utilisable pour un conteneur avec l'option *--memory*.

En parallèle à cette option, le conteneur peut être autorisé à utiliser de l'espace disque dur pour l'excès de mémoire (swap), pour autant que le système d'exploitation le permette (ce qui n'est pas le cas par défaut sur notre serveur Linux).

Il est possible également de spécifier une limite inférieure à *--memory (--memory-reservation*) qui sera prise en compte par Docker si la quantité de mémoire disponible sur une machine est trop faible. Ceci permet au conteneur d'éviter si possible d'utiliser plus de mémoire que celle définie par cette limite.[37]

Si vous souhaitez plus d'informations, vous pouvez consulter le site https://docs.docker.com/config/containers/resource_constraints/#memory.

[37] https://docs.docker.com/config/containers/resource_constraints/#memory, consulté le 10.11.2018

5.4.1. Définition d'un quota dans le service Web

Dans notre service Web JAVA, nous définissons un quota de RAM comme illustré dans la figure 24 (par exemple $5*10^6$, soit 4.7 MB en réalité après conversion en binaire).

```
//Ajout d'une limite RAM et CPU disponible pour la tâche
//(qui exécute le conteneur) lancée par le service
final ResourceRequirements resourRequi = ResourceRequirements.builder()
        .limits(Resources.builder().memoryBytes(5000000L).build())
        .build();

//Conditions de redémarrage de la tâche dans le service
//(ici, redémarrer en cas d'échec, jusqu'à 5 tentatives max)
final RestartPolicy restartPolicy = RestartPolicy.builder()
        .condition(RestartPolicy.RESTART_POLICY_ON_FAILURE)
        .maxAttempts(5)
        .build();

//Définition de la tâche à démarrer dans le service
final TaskSpec taskSpecs = TaskSpec.builder()
        .containerSpec(containerSpecs)
        .restartPolicy(restartPolicy)
        .placement(placeSpecs)
        .resources(resourRequi)
        .build();
```

Figure 26 Spécification de limite mémoire (dans service Web)

5.4.2. Test d'exécution avec limite RAM

5.4.2.1. Fonctionnement normal

Une fois le quota défini dans le service Web, nous pouvons effectuer des tests pour comprendre le comportement des conteneurs déployés avec cette limite.

Nous constatons qu'en créant un service avec une image qui contient une interface Web simple et qui utilise normalement environ 16 MB fonctionne (malgré quelques légères latences) et, avec cette limite, elle plafonne à 4.65 MB (Figures 27 et 28).

```
[redacted]:~$ docker run engel1605/get-started:part2
 * Serving Flask app "app" (lazy loading)
 * Environment: production
   WARNING: Do not use the development server in a production environment.
   Use a production WSGI server instead.
 * Debug mode: off
 * Running on http://0.0.0.0:80/ (Press CTRL+C to quit)

File  Edit  View  Search  Terminal  Help
CONTAINER ID        NAME                CPU %               MEM USAGE / LIMIT
 MEM %               NET I/O             BLOCK I/O           PIDS
5c0f3a93857d        angry_pike          0.03%               16.93MiB / 3.853GiB
 0.43%               796B / 0B           664kB / 0B          1
```

*Figure 27 Exécution image Docker **sans** limite RAM*

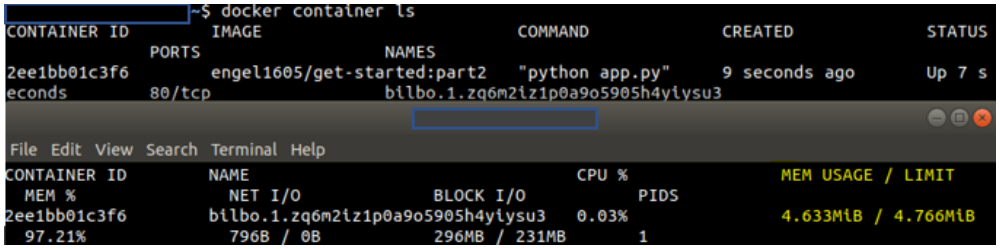


*Figure 28 Exécution image Docker **avec** limite RAM*

5.4.2.2. Stress test

Les *stress tests* permettent de tester les performances d'un système ou d'un logiciel en augmentant l'utilisation de la mémoire, des processeurs et autres ressources.

Dans notre cas, nous utilisons l'outil *stress* sur un conteneur avec une image Docker Ubuntu qui contient un serveur Web Apache et dans laquelle nous avons installé cet outil. Le quota de RAM est fixé à 256 MB. La figure 29 illustre la tâche démarrée par notre service créé (*stressram*), ainsi que l'utilisation actualisée de la mémoire par le conteneur et ses processus qui sont en cours d'exécution.

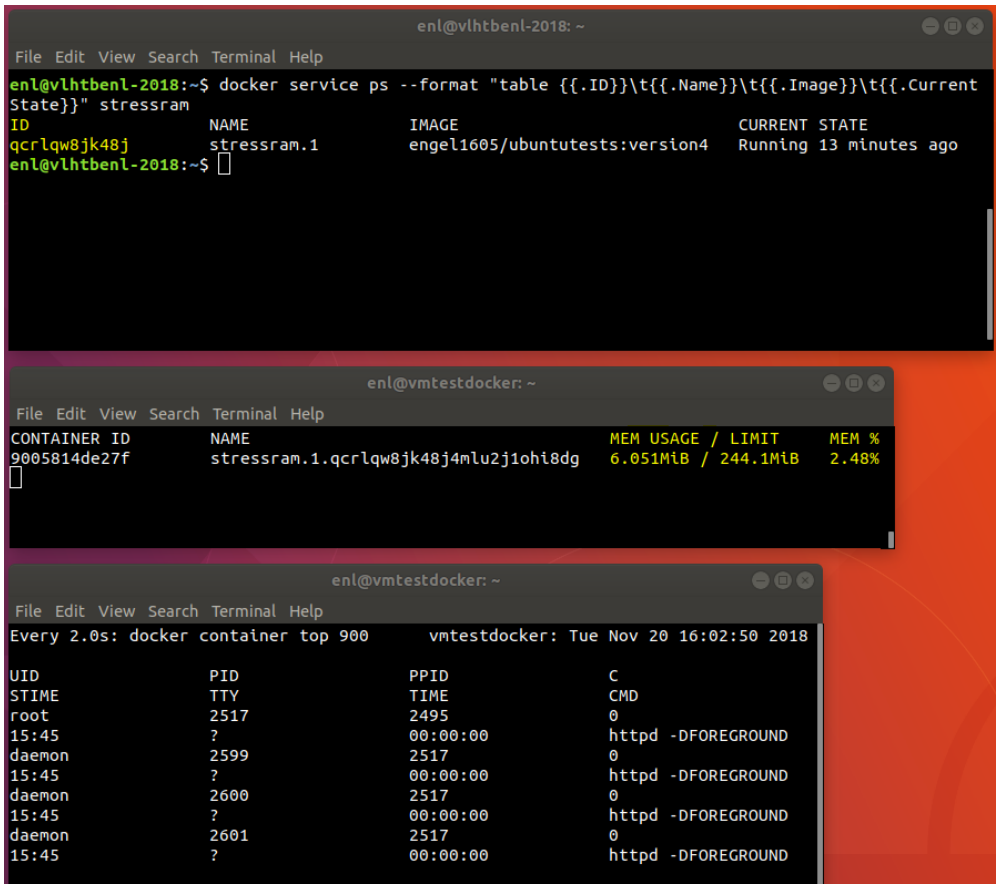


Figure 29 Service Docker avec sa tâche, utilisation RAM et ses processus

Nous démarrons à présent un processus *stress* sur ce conteneur avec la commande *docker exec <id du conteneur> stress -m 2G* (pour forcer 2 GB d'utilisation de RAM). Nous constatons dans la figure 30 que la tâche « *qcrlqw8jk48j* » et son conteneur « *9005814de27f* » illustrés ci-dessus sont arrêtés (erreur) et remplacés par de nouveaux.

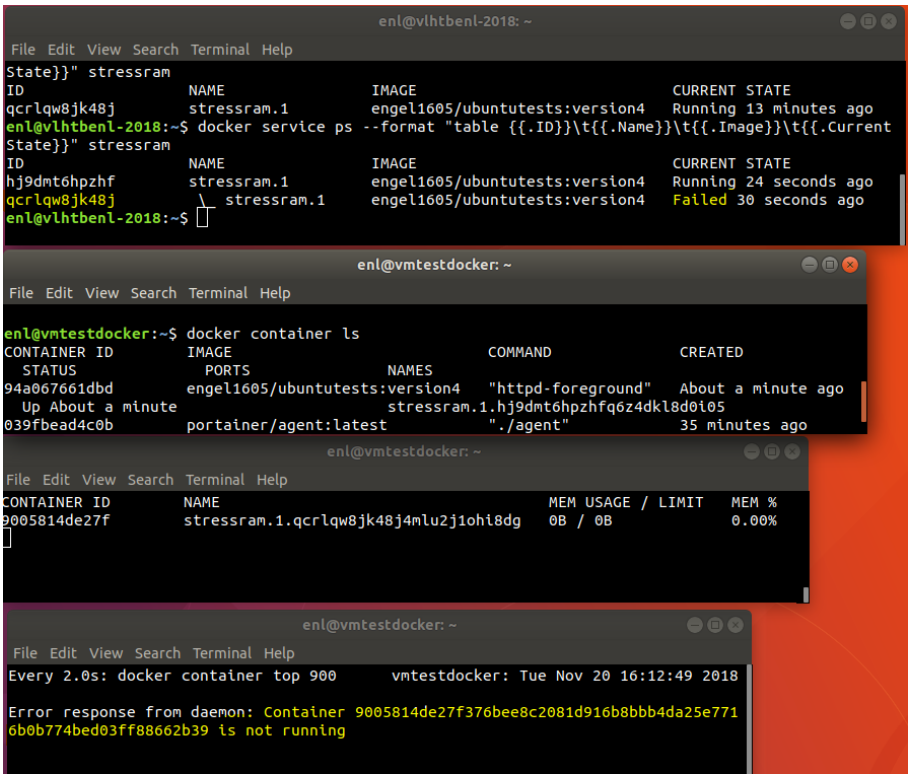


Figure 30 Résultat stress test limite RAM

5.5. Quotas CPU

Pour ce qui est du quota CPU, nous pouvons utiliser le *CFS scheduler* (*Completely Fair Scheduler*) ou *Linux real-time scheduler*, qui permettent au noyau (*kernel*) Linux d'une machine de prioriser le traitement de tâches pour chaque processeur.

5.5.1. CFS scheduler

Le *CFS scheduler* permet un partage équitable des ressources de processeurs et de la période mise à disposition des tâches. Par exemple, si nous avons quatre processus dans une queue, chacun d'eux auraient droit à 25% de temps du processeur concerné.[38]

Le principe de base du côté de l'utilisateur est simple. Si nous utilisons *docker run*, nous définissons le nombre de processeurs qu'un conteneur peut utiliser avec l'option *–cpus=XX* (recommandé à partir de Docker 1.13). Docker calcule alors la période et le quota.[39]

[38] https://www.youtube.com/watch?v=MkJfuI5_hjc, consulté le 15.11.2018

[39] https://docs.docker.com/config/containers/resource_constraints/#configure-the-default-cfs-

5.5.2. Real-time

Les processus *real-time* nécessitent un temps de traitement court avec un minimum de latence (par exemple : des applications de traitement vidéo ou audio). Ils ont ainsi la priorité dans une queue d'un processeur.

Ubuntu 18.04 (Bionic) ne prend pas en charge le *real-time* par défaut (comme indiqué par Docker lorsque nous essayons d'utiliser cette fonctionnalité en exécutant *docker run*).[40] Il est donc nécessaire d'installer et compiler un noyau (*kernel) real-time* pour effectuer une comparaison avec le *CFS*.

5.5.3. Définition d'un quota CPU (service Web)

En ce qui concerne notre service Web, une limite en *nanoCPU* (10^{-9} nombre de processeurs) est définie au niveau de la tâche du service Docker créé (Figure 30). Le quota est calculé par Docker de la manière suivante : Quota = nanoCPUs * périodeCPU / 10^{-9}. Dans notre situation, nous définissons 1.5 * 10^{9} nanoCPU, donc 1.5 processeurs.

```
//Ajout d'une limite RAM et CPU disponible pour la tâche
//(qui exécute le conteneur) lancée par le service
final ResourceRequirements resourRequi = ResourceRequirements.builder()
        .limits(Resources.builder().memoryBytes(5000000L).nanoCpus(1500000000L).build())
        .build();

//Conditions de redémarrage de la tâche dans le service
//(ici, redémarrer en cas d'échec, jusqu'à 5 tentatives max)
final RestartPolicy restartPolicy = RestartPolicy.builder()
        .condition(RestartPolicy.RESTART_POLICY_ON_FAILURE)
        .maxAttempts(5)
        .build();

//Définition de la tâche à démarrer dans le service
final TaskSpec taskSpecs = TaskSpec.builder()
        .containerSpec(containerSpecs)
        .restartPolicy(restartPolicy)
        .placement(placeSpecs)
        .resources(resourRequi)
        .build();
```

Figure 31 Spécification de limite CPU (dans service Web)

scheduler, consulté le 11.11.2018

[40] https://ubuntuforums.org/showthread.php?t=2396610, consulté le 15.11.2018

5.5.4. Tests d'exécution (CFS scheduler)

5.5.4.1. Fonctionnement normal

En créant un service Docker depuis le Web, nous constatons, en inspectant le conteneur, que le quota de CPU a été calculé et attribué (Figure 31).

```
enl@vmtestdocker:~$ docker container inspect --format "ID: {{.ID}} Name: {{
.Name}} CPUQuota: {{.HostConfig.CPUQuota}} CPUPeriod: {{.HostConfig.CPUPeri
od}}" 3da
ID: 3da74e497d589a4d3e353d0b1de6bcbfe004dc05321eae1872d8c4cf71d081a9 Name:
/stresscpu.1.ochwhos139bweoozfjr1s16p7 CPUQuota: 150000 CPUPeriod: 100000
enl@vmtestdocker:~$
```

Figure 32 Propriétés conteneur: CPUQuota et CPUPeriod

5.5.4.2. Stress test

Nous exécutons ensuite sur ce container défini un test de performance en utilisant le même outil *stress* que pour les tests de quotas RAM avec la commande *docker exec <id du conteneur> stress -c 4* (pour utiliser 4 cpus).

La figure 32 illustre la tâche démarrée par notre service créé (*stresscpu*), ainsi que l'utilisation actualisée de la mémoire par le conteneur et ses processus qui sont en cours d'exécution. Comme nous pouvons le voir, l'utilisation de CPU augmente jusqu'à environ 100% d'utilisation.

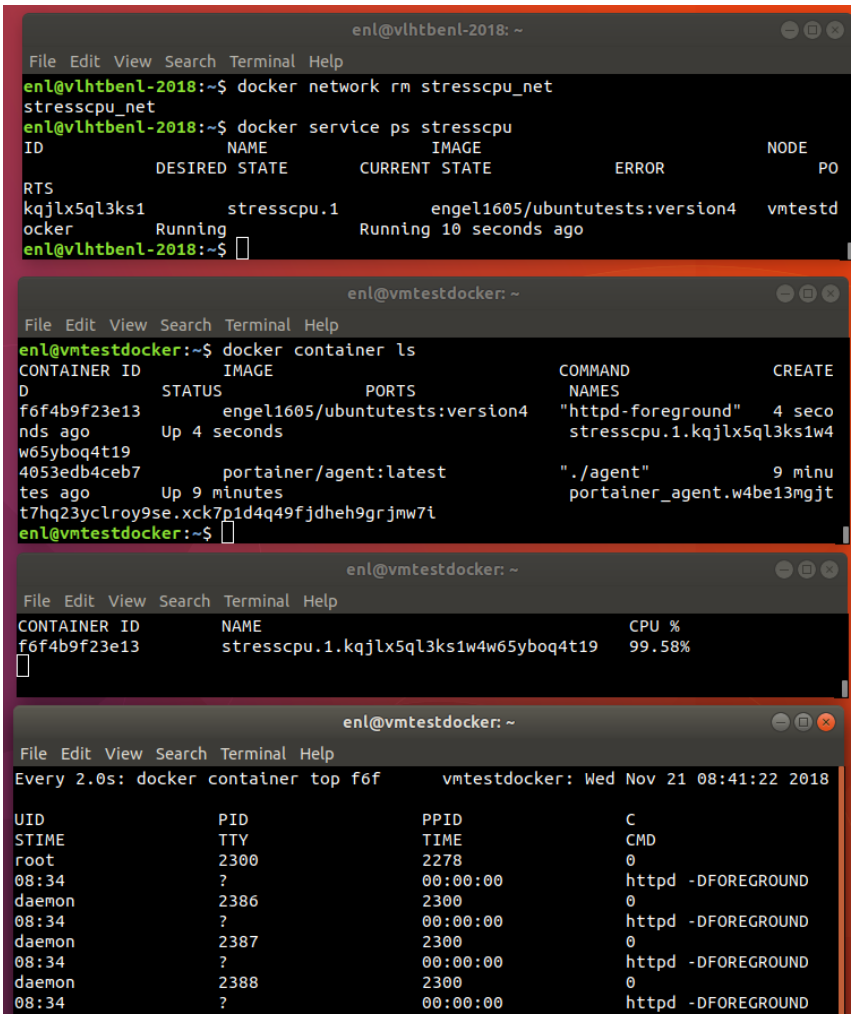


Figure 33 Résultat stress test limite CPU

5.5.5. Conclusion

Etant donné le manque de temps pour la compilation d'un noyau *real-time*, nous choisissons de conserver et tester le *CFS*, qui convient pour effectuer une analyse de données.

5.6. Réseau Docker pour conteneurs

5.6.1. Définitions

Docker fournit 5 pilotes réseau par défaut [41] [42]:

- ***Bridge*** : Il crée un réseau interne isolé au sein d'un seul équipement pour permettre aux conteneurs de communiquer entre eux. Docker gère le lien (pont) entre ce réseau et celui de la machine hôte. Si besoin, des ports externes peuvent être exposés au conteneur. Ce pilote simple est utilisé si aucun autre n'est précisé dans l'exécution d'un service ou conteneur.
- ***Overlay*** : Ce type de réseau s'étend à toutes les machines dans un *Swarm* et permet une communication entre conteneurs et une distribution équilibrée sur différentes machines, de manière encryptée si souhaité. Lorsqu'un nouveau réseau utilisant ce pilote est créé, il sera répliqué également sur les autres nœuds, lorsqu'une tâche fera appel à ce réseau.[43]
- ***Host*** : Ce pilote réseau lie le réseau au sein d'un conteneur directement à celui de la machine hôte. Il est donc utilisé uniquement en cas de publication d'un port (par ex : 80 pour Web).
- ***Macvlan*** : Docker attribue une adresse réseau MAC à un conteneur relié directement au réseau physique. Celui-ci est alors visible en tant que périphérique.
- ***None*** : Le container est complétement isolé au niveau du réseau, aucune interface n'est configurée.

[41] https://blog.docker.com/2016/12/understanding-docker-networking-drivers-use-cases/, consulté le 08.11.2018

[42] https://success.docker.com/article/networking, consulté le 08.11.2018

[43] https://docs.docker.com/network/network-tutorial-overlay/, consulté le 16.11.2018

5.6.2. Création d'un nouveau réseau

En ligne de commande Docker, nous créons ce réseau en entrant la commande *docker network create --driver overlay --internal <nomDuService>_net* .

Dans notre service Web, nous définissons une variable (ex : *netConfig*) qui contient le nom du pilote à utiliser, l'option *internal* pour restreindre l'accès vers l'extérieur du réseau et le nom du réseau souhaité, puis nous envoyons la requête de création sur le client Docker. (Figure 33)

```
//Création d'un nouveau réseau Docker
NetworkConfig netConfig = NetworkConfig.builder()
        .driver("overlay")
        .internal(true)
        .name(serviceName + "_net")
        .build();
docker.createNetwork(netConfig);
```

Figure 34 Création réseau Docker (dans le service Web)

5.6.3. Définition du réseau à utiliser (service Web)

En ligne de commande Docker, nous pouvons rattacher un service à un réseau existant avec l'option *docker service create* ***--network <nom du réseau>***.

Depuis notre service Web, nous écrivons une variable (ex : *network*) dans laquelle la méthode *target* permet d'indiquer l'indiquer le nom du réseau à utiliser (ici, nous reprenons le nom du réseau créé dans le point précédent) . Puis nous ajoutons cette variable dans les spécificités du service. (Figure 34)

```
//Spécification du réseau lié au service
final NetworkAttachmentConfig network = NetworkAttachmentConfig.builder()
        .target(netConfig.name())
        .build();

//Définition du service avec son nom, la tâche, le mode (répliqué 1 fois
//=> 1 seul conteneur) et le réseau
final ServiceSpec serviceSpecs = ServiceSpec.builder()
        .name(serviceName)
        .taskTemplate(taskSpecs)
        .mode(ServiceMode.withReplicas(1L))
        .networks(network)
        .build();
```

Figure 35 Définition réseau Docker à utiliser (dans service Web)

5.6.4. Tests de comparaison

Le but dans le cadre de ce projet est d'isoler totalement un conteneur au niveau du réseau, mis-à-part la communication nécessaire avec le nœud *manager* du *Swarm* pour le bon fonctionnement.

Nous pouvons déjà exclure le type *Host* dans ce projet, puisque le réseau du conteneur sera lié en direct à la machine où celui-ci est exécuté. De plus, il nous est impossible d'attacher un service dans le réseau *ingress* (*overlay* par défaut créé par Docker pour la communication entre conteneurs et le *Swarm*).

Les tests ci-dessous sont exécutés en créant un service qui exécute une image basée sur *httpd.* Sur le container démarré à partir de cette image, nous effectuons des commandes *ifconfig*, *ping* et *traceroute*.

Critères d'évaluation :

- Adresse réseau attribuée au conteneur
- Traffic réseau vers un autre conteneur
- Accès aux deux points de montage NAS sur l'hôte
- Traffic réseau vers l'hôte physique
- Traffic réseau vers une autre machine dans le même réseau physique
- Communication avec le cœur du Swarm
- Traffic réseau externe

Types de réseau évalués :

- *bridge* : *bridge* par défaut créé par Docker
- *docker_gwbridge* : *bridge* qui connecté aux interfaces réseau de la machine hôte mais aussi au réseau *ingress*
- *ovNet* : *overlay* de base (créé par l'auteur sans option supplémentaire)
- *mcNet* : *macvlan* de base (créé par l'auteur sans option supplémentaire)
- *intOvNet* : *overlay* créé par l'auteur avec l'option *--internal* pour bloquer le trafic hors du réseau et *--opt encrypted=true* pour permettre un trafic encrypté
- *<nomService>_net* : réseau basé sur *overlay* créé lors du lancement d'un nouveau service Docker via spotify/Docker-client (dans notre service Web). Le nom du réseau est basé sur celui du service Docker

Réponses possibles :

- Oui
- Non
- Non applicable (N/A)

	Adresse réseau	Autre conteneur	Montage NAS	Hôte physique	Autre machine	Cœur du Swarm	Réseau externe
bridge	Oui	Oui	Oui	Oui	Oui	Oui	Oui
docker_gwbridge	N/A	N/A	N/A	N/A	N/A	N/A	N/A
ovNet	Oui	Oui	Oui	Oui	Oui	Oui	Oui
mcNet	N/A	N/A	N/A	N/A	N/A	N/A	N/A
intOvNet	Oui	Oui	Oui	Non	Non	Oui	Non
<service>_net	Oui	Non	Oui	Non	Non	Oui	Non

Tableau 2 Tests comparaison réseaux Docker

Constats lors des tests :

- *ovNet* : deux interfaces réseau sont créées dans un conteneur : une liée au réseau *ovNet* et une autre connectée au réseau *docker_gwbridge*.
- *docker_gwbridge* et *mcNet* : nous constatons qu'il faut utiliser un réseau qui s'étend au *Swarm* (ex : avec le pilote *overlay*).

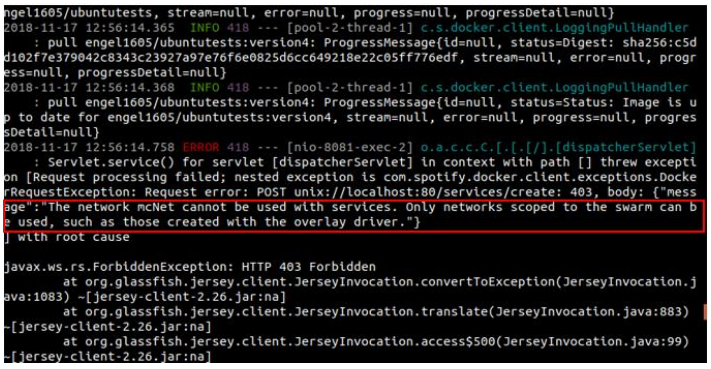


Figure 36 Erreur: uniquement réseau de scope Swarm pour services Docker

- *intOvNet* : contrairement à *ovNet*, il y a une seule interface, connectée au réseau *intOvNet*, grâce à l'option *--internal*

- <*service*>*_net* : cette possibilité crée un réseau par service démarré, ce qui signifie que nous avons une liste avec un certain nombre de réseaux différents.

5.6.5. Conclusion

A partir des tests réalisés, nous constatons que la possibilité optimale pour permettre une communication réseau uniquement avec le cœur *Swarm* et de pouvoir accéder aux points de montage sur la machine hôte est de créer un réseau basé sur *overlay* avec l'option *internal*, lors du lancement d'un nouveau service. Il y a donc un réseau par service.

5.7. *Scheduler* / queue Docker swarm

5.7.1. Processus de gestion des services et tâches

Le *Swarm* utilise un algorithme pour gérer les services et établir des tâches liées à ceux-ci. Ce mécanisme est illustré dans la figure 36 ci-dessous.

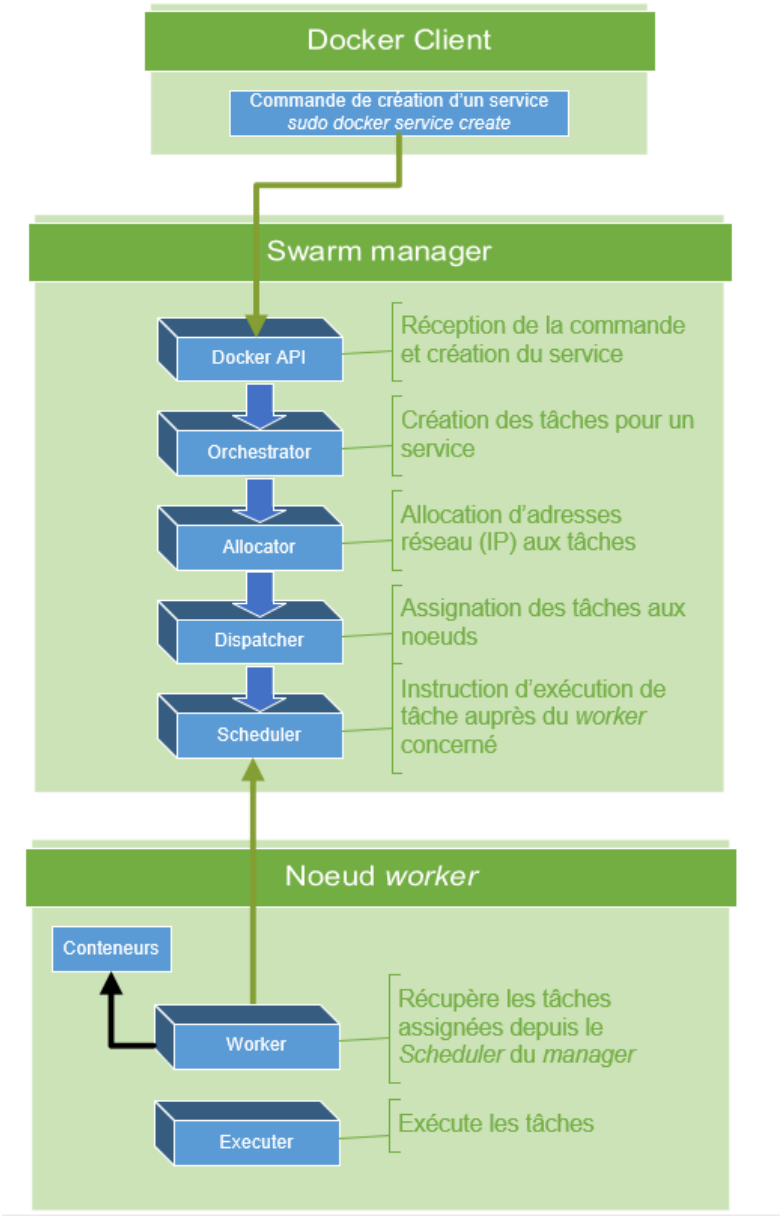


Figure 37 Traitement d'un service (scheduler Swarm)[44]

[44] Réalisé par l'auteur de ce rapport depuis https://docs.docker.com/engine/swarm/how-swarm-mode-works/services/, consulté le 9.11.2018

5.7.2. Filtres

Ce point concerne les filtres qui peuvent être utilisés par le *Swarm*. Il est basé sur une vidéo réalisée par Docker. [45]

En premier, le planificateur (*scheduler*) du gestionnaire de *Swarm* se base d'abord sur des filtres pour attribuer les tâches.

Dans le cas où un conteneur (dans une tâche) nécessite un certain port réseau, le planificateur va ignorer les nœuds du *Swarm* dans lesquels ce port est déjà utilisé par un autre conteneur.

Ensuite, si par exemple Docker Engine sur un nœud n'est pas disponible (en pause, injoignable ou autre), celui-ci est également ignoré.

Par ailleurs, le planificateur peut se baser sur les contraintes définies au sein du service Docker créé pour distribuer les tâches sur les nœuds. Ces contraintes peuvent être un identifiant de nœud, un nom d'hôte, une adresse réseau IP, le rôle (*manager* ou *worker*), le système d'exploitation, l'architecture (64 bits, 32 bits) ou des étiquettes (telles que *default, HighRAM, HighCPU* que nous utilisons dans ce projet).

Enfin, des affinités peuvent être définies pour un conteneur, si nous voulons le créer en parallèle à un autre, pour autant qu'il n'y ait pas de contraintes établies.

5.7.3. Stratégies

Ce point concerne les stratégies qui peuvent être utilisées par le Swarm. Il est basé sur la même vidéo que les filtres (point 5.7.2).[46]

En deuxième étape, après application des filtres, le planificateur va se baser sur une des trois stratégies possibles :

- ***Binpack*** : le gestionnaire du *Swarm* attribue les conteneurs d'abord sur le premier nœud. Quand celui-ci est plein, il passe au suivant et ainsi de suite

[45] https://www.youtube.com/watch?v=7B_bX3czq-Y, consulté le 9.11.2018

[46] https://www.youtube.com/watch?v=7B_bX3czq-Y, consulté le 9 novembre 2018

- ***Random*** : les conteneurs sont distribués sur les nœuds de façon aléatoire. Ceci est d'abord utilisé pour des tests (*debugging)*.[47]
- ***Spread*** : stratégie par défaut dans un *Swarm*. Les conteneurs sont distribués de manière « équitable » entre les machines, en fonction des ressources disponibles.

5.7.4. Conclusion

Dans le cadre de ce travail, nous utilisons la stratégie *Spread*, qui correspond à nos besoins, étant donné également que nous voulons établir des quotas de RAM et de CPU. De plus, nous utilisons les filtres de nœuds (contraintes) pour distribuer les conteneurs sur les bons nœuds en fonction des capacités des ressources.

[47] https://github.com/docker/docker.github.io/blob/master/swarm/scheduler/strategy.md, consulté le 24.11.2018

5.8. Interface web cliente

Pour finir, une interface web est mise en place afin de permettre aux clients (chercheurs dans notre cas) de créer un service Docker dans lequel un conteneur exécutera l'image qu'ils définissent, sur une machine appartenant au *Swarm* en fonction du type voulu (dans notre cas *HighRAM*, *HighCPU* ou par défaut). Il est également possible de lister les services initialisés sur le serveur.

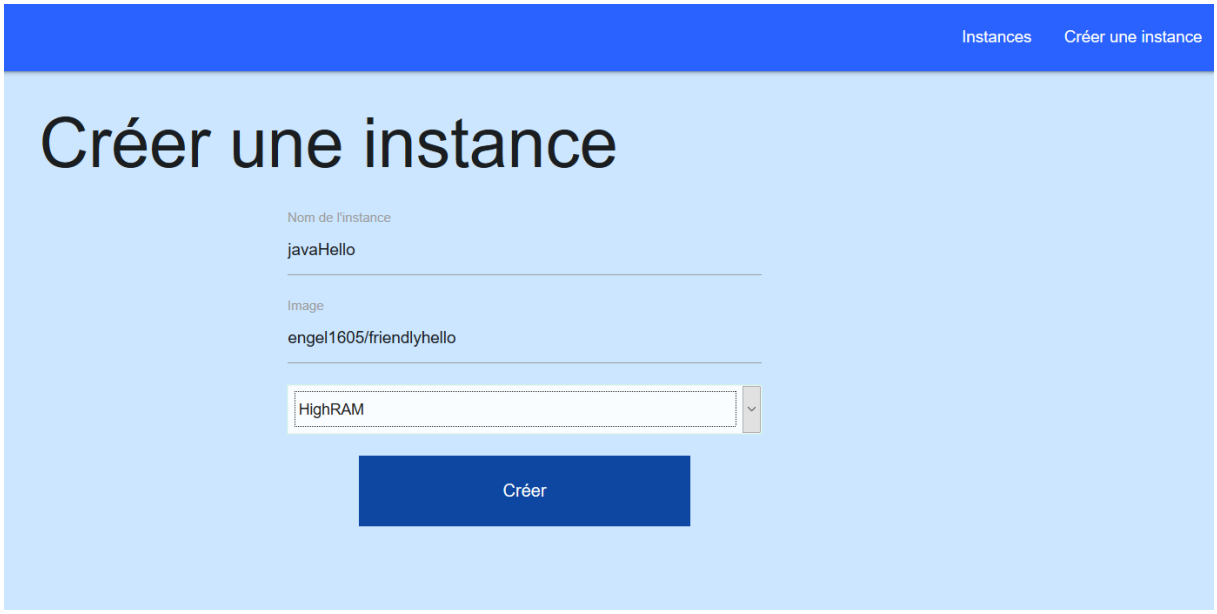


Figure 38 Création d'instance (Interface web cliente)

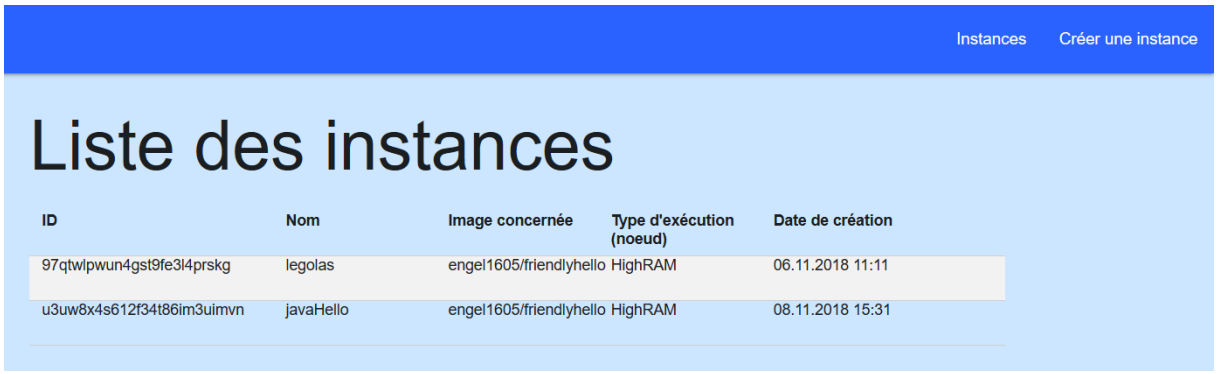


Figure 39 Liste des instances (Interface web cliente)

5.9. Registre Docker interne privé

Un objectif secondaire (*nice-to-have*) du cahier des charges est de mettre en place un registre Docker privé accessible en interne. Nous nous basons pour cela sur la documentation de Docker.[48]

Étant donné que nous souhaitons accéder à ce nouveau registre depuis d'autres machines du *Swarm*, la première étape consiste à générer un certificat de sécurité avec la commande *sudo openssl req -x509 -nodes -days 365 -newkey rsa:2048 -keyout /etc/ssl/private/<nom de la clé>.key -out /etc/ssl/certs/<nom du certificat>.crt.*[49] Le certificat doit ensuite être copié sur chaque hôte Docker dans */etc/docker/certs.d/<nom du registre>:5000/.* Dans ce travail, le certificat est auto-signé (*self-signed*), mais il faut éviter cela en production et utiliser un certificat validé par une autorité de confiance (*Certificate Authority*).

La deuxième étape consiste à ajouter un/des login(s) et mot(s) de passe pour restreindre l'accès au registre à travers l'authentification basique (un objectif est ensuite d'utiliser une authentification avec OpenID Connect et Keycloak, mais il ne sera pas implémenté dans ce travail). Pour ce faire, il faut exécuter la commande *docker run --entrypoint htpasswd registry :2 -Bbn <nom utilisateur> <mot de passe> > <chemin souhaité>/htpasswd*, puis arrêter ce conteneur après.

La troisième étape consiste à créer le fichier *docker-compose.yml* (Figure 40), puisque nous déployons le registre en tant que service Docker. Nous spécifions l'emplacement du certificat et de la clé générés, ainsi que celui du fichier créé pour l'authentification de base.

[48] https://docs.docker.com/registry/deploying/, consulté le 25.11.2018

[49] https://www.digitalocean.com/community/tutorials/how-to-create-a-self-signed-ssl-certificate-for-apache-in-ubuntu-16-04 (Etape 1), consulté le 25.11.2018

```
version: "3"

services:
  registry:
    image: registry:2
    deploy:
      replicas: 1
      placement:
        constraints:
          - node.role == manager
    ports:
      - 5000:5000
    environment:
      REGISTRY_HTTP_TLS_CERTIFICATE: /certs/docker-selfsigned.crt
      REGISTRY_HTTP_TLS_KEY: /certs/docker-selfsigned.key
      REGISTRY_AUTH: htpasswd
      REGISTRY_AUTH_HTPASSWD_PATH: /auth/htpasswd
      REGISTRY_AUTH_HTPASSWD_REALM: Registry Realm
    volumes:
      - /dockerRegistry/certs:/certs
      - /dockerRegistry/auth:/auth
```

Figure 40 Fichier Docker compose pour création registre interne

Pour finir, nous déployons le service sur le Swarm avec *docker stack deploy -c docker-compose.yml <nom du service>*. Ensuite, nous nous connectons et nous testons le chargement d'une image sur ce registre (Figure 41 et 42). Attention, l'image doit posséder une étiquette (*tag*) du type *<nom d'hôte où se trouve le registre> :<port>/<nom d'image>* (ex : *localhost :5000/my-apache*) pour que Docker sache qu'il doit l'envoyer sur un registre différent de Docker Hub.[50]

```
[redacted]:~$ docker login [Nom hôte registre] :5000
Username: bilbo
Password:
Login Succeeded
```

Figure 41 Login sur registre Docker interne

```
[redacted]:~$ docker push [redacted]:5000/friendlyhello
The push refers to repository [[redacted]:5000/friendlyhello]
93c5dfec3025: Layer already exists
13db37a527f2: Pushed
7dbf478d8826: Layer already exists
705be221ae37: Layer already exists
9803ecbfb7a5: Layer already exists
b71576f95f0e: Pushed
53e7bc96c800: Pushed
3d1f1997a0df: Pushed
eea2bb10fa79: Pushed
latest: digest: sha256:0d6456d35e189e3efa32714efc8bb4266fa560b6054ba7abff
9d99fa size: 2208
```

Figure 42 Upload image Docker sur registre interne

[50] https://docs.docker.com/registry/deploying/#copy-an-image-from-docker-hub-to-your-registry, consulté le 25.11.2018

6. Conclusion

6.1. Bilan technique

En l'état actuel, Docker est installé sur nos machines et le mode *Swarm* est fonctionnel.

De plus, notre service Web permet d'exécuter des conteneurs à partir d'une image Docker, avec les paramètres souhaités (quotas RAM, quotas CPU, réseau isolé, points de montage et conditions de redémarrage). De plus, une fonction permet de lister les services déployés et leurs propriétés.

L'interface Web cliente, qui sera utilisée lors de la défense de ce travail, est déjà implémentée. Nous pouvons créer des services Docker et afficher ceux qui existent. De plus, nous pouvons utiliser un outil graphique pour visualiser de manière plus claire l'utilisation des ressources par les conteneurs Docker (par exemple : Portainer) ou afficher ces mesures directement dans notre interface.

6.2. Améliorations futures / recommandations

Par la suite, nous devons encore continuer le développement de l'interface Web de démonstration. Il faut notamment afficher un message d'erreur si, par exemple, le chercheur a entré le nom d'une image Docker inexistante. Ces messages seront liés à ceux retournés par le service Web en cas d'erreur.

Le service Web est disponible à travers le protocole de communication HTTP. C'est pourquoi nous devons encore implémenter le protocole HTTP**s** pour améliorer la sécurité.

La structure du code du service Web peut être éventuellement améliorée en créant des fonctions séparées qui seraient ensuite appelées dans la fonction de création de service Docker, pour améliorer la lisibilité.

En ce qui concerne les risques liés à Docker et l'exécution de code arbitraire qui n'ont pas été adressés dans ce travail :

1) un moyen pour limiter l'utilisation de l'espace de stockage doit encore être trouvé
2) il faut être attentifs et appliquer des *best practices* de sécurité supplémentaires qui sont affichées par l'outil Docker bench security.
3) il faut encore mettre en place un outil (par exemple Anchore engine) pour analyser les

images Docker avant leur déploiement.

4) le module de sécurité Linux *AppArmor* peut être configuré de manière plus précise pour appliquer des profils d'isolation des conteneurs.
5) il faut activer *DOCKER_CONTENT_TRUST* pour autoriser uniquement le téléchargement d'images de confiance (« signées »).
6) les conteneurs devraient être isolés à travers des *user namespaces*. Il faut regarder s'il est possible de le paramétrer dans le service Web pour les nouveaux conteneurs créés.
7) il faudrait tester *ebtables* pour filtrer le trafic et bloquer en particulier les attaques *ARP spoofing*
8) il faut mettre un système d'authentification sur l'interface Web cliente.

6.3. Problèmes rencontrés

En ce qui concerne le service Web développé, nous avons dû ajouter des librairies dont il dépend, car une erreur inconnue est survenue lors de tentatives de déploiement. De plus, nous avons constaté que les conteneurs arrêtés automatiquement pour cause de dépassement du quota de mémoire, ce qui est dû aux conditions de redémarrage liées au service Docker que nous créons via le service Web. Ces problèmes ont néanmoins été résolus relativement rapidement.

De plus, nous n'avions pas accès aux volumes contenants des données de tests sur le NAS, car la connexion était bloquée par défaut dans le pare-feu de la HES-SO Valais. Le Service informatique a donc effectué les opérations nécessaires.

6.4. Respect du cahier des charges

Le *product backlog* joint en annexe II reprend les objectifs du cahier des charges (Annexe I). Les points principaux (*must have*) ont été en grande partie réalisés dans ce travail :

- ✓ Etat de l'art autour de Docker : **réalisé**
- ✓ Analyse détaillée des risques (infrastructure, sécurité et implications légales) : **réalisé**
- ✓ Mise en place d'une infrastructure Docker locale avec mode *Swarm* : **réalisé**
- ✓ Lancement de conteneurs via un service Web qui utilise spotify/docker-client : **réalisé**
- ✓ Blocage du trafic réseau externe : **réalisé**
- ✓ Définition de quotas RAM et CPU dans le service Web : **réalisé en partie, nous n'avons pas pu tester un CPU *realtime***

- ✓ Explication / utilisation du *Swarm scheduler* : **réalisé**
- ✓ Création d'une interface web basique pour démonstration : **réalisé**

Comme dit dans le point 6.1, nous améliorerons encore l'interface web et nous utiliserons un moyen graphique pour visualiser les ressources utilisées.

Etant donné qu'il restait du temps, nous avons également installé un registre Docker interne sur notre serveur, avec certificats et authentification de base (point *nice-to-have* du cahier des charges).

Nous avons essayé de trouver le moyen de définir une durée de vie maximale sur un conteneur (objectif *nice-to-have* du cahier des charges). Malheureusement, nous n'avons trouvé aucune possibilité qui satisfasse ce point.

Enfin, par manque de temps, nous n'avons pas pu réaliser l'analyse d'alternatives à Docker Hub, ni la mise en place d'une procédure de « Conditions d'utilisation » des données de patients (objectif *nice-to-have* du cahier des charges).

6.5. Conclusion personnelle

La réalisation de ce travail a été très enrichissante pour moi. J'ai choisi ce thème, car il s'agit d'une application de l'informatique dans le domaine médical et je trouve que la technologie peut apporter d'énormes améliorations dans ce secteur.

De plus, ce travail fait partie d'un projet réalisé par des membres de l'institut Informatique de Gestion de Sierre et c'est très gratifiant de savoir que mes recherches et mon développement seront repris dans ce projet.

Ensuite, j'avais beaucoup entendu parler de Docker et je l'avais utilisé de façon basique dans un cours durant mon cursus, sans le connaître . Ainsi, dans ce travail, j'ai pu comprendre les capacités et fonctionnalités de cet outil puissant et efficace. J'espère pouvoir reprendre ces connaissances acquises et réutiliser Docker dans mon futur professionnel.

Pour finir, après avoir réalisé plusieurs projets en équipe, j'ai appris à travailler sur un projet complet de façon autonome, en donnant le meilleur de moi-même, tout en communiquant avec mon professeur, Dr. Henning Müller, et le répondant technique, M. Ivan Eggel.

Références

Chelladhurai, J. S., Singh, V., & Raj, P. (2017). *Learning Docker Ed. 2*. Packt Publishing. Récupéré sur http://univ.scholarvox.com/book/88842853.

Mckendrick, R., & Gallagher, S. (2017). *Mastering Docker Ed. 2*. Packt Publishing, Récupéré sur http://univ.scholarvox.com/book/88842932#.

Dua, R., Raja, A. R., & Kakadia, D. (2014). Virtualization vs Containerization to Support PaaS. *2014 IEEE International Conference on Cloud Engineering* (p. 610-614). Récupéré sur https://ieeexplore.ieee.org. (10.1109/IC2E.2014.41).

Eggel, I., Schaer, R., & Muller, H. (Juin 2018). Distributed container-based evaluation platform for private/large datasets. *2018 17th International Symposium on Parallel and Distributed Computing* (p. 93-100). Récupéré sur https://ieeexplore.ieee.org. (10.1109/ISPDC2018.2018.00022).

Pochon, Y. (2016). *Utilisation de Docker dans la science des données* (Mémoire de bachelor : Haute Ecole de Gestion & Tourisme). Récupéré sur RERO DOC. (oai:doc.rero.ch:20170412124639-LP).

Laghzaoui, K. (2017). *Docker®* (Mémoire de bachelor : Haute école de gestion de Genève). Récupéré sur RERO DOC. (oai:doc.rero.ch:20180126082036-BE).

Robail, Y. (2018). Mitigating Docker Security Issues. *Cryptography and Security*. Récupéré sur arXiv (arXiv:1804.05039).

Engel, L. (2018). *Intégration du RGPD et de la nouvelle LPD auprès de la HES-SO Valais-Wallis*.

Zuber, J. (2013). *Guide de présentation et réalisation des travaux écrits*. Récupéré sur http://intranet.hevs.ch/fr-fr/formation/informatique-de-gestion/travail-de-bachelor.

Annexe I : Cahier des charges

Présentation

I. Rôles

- Etudiant :
  - **Lionel Engel**

- Professeur :
  - **Dr. Henning Müller**

- Support technique :
  - **Ivan Eggel**

Remarque générale

Une bonne partie des spécifications originales ont déjà été implémentées en interne, car aucun étudiant n'avait choisi ce thème de Travail de Bachelor proposé au printemps 2018. Le but de ce travail est de trouver un concept d'implémentation pour les éléments qui n'ont pas encore été réalisés, qui pourra ensuite être utilisé par l'équipe du projet global.

Degré de difficulté

Moyen - élevé

Implémentation actuelle

- Mise en place de *Docker swarm* (cluster Docker)
- Interface web pour lancer des containers Docker qui contiennent du code d'analyse des données
- Liaison du container avec des données internes (*bind mount*)
- Choix d'environnement d'exécution par l'utilisateur (ex : « High CPU », « High RAM », « GPU »)
- Stockage des résultats du code d'analyse de données
- Évaluation des résultats d'analyses

Objectifs

I. « Must have » (obligatoires)

- Etat de l'art autour de Docker
  - Concepts
  - Versions
    - Community
    - Enterprise (Basic, Standard and Advanced)
  - Standards et conformités
  - Outils et méthodes disponibles
    - Swarm mode
    - Docker Hub
    - …
- Analyse détaillée des risques d'exécution de code étranger arbitraire (contenu dans container Docker) sur des machines internes en lien avec des données sensibles
  - Infrastructure
  - Sécurité (*sandboxing*, failles, corruption de données, fuite de données, …)
  - Légal (LPD[1], P-LPD[2], RGPD[3])
- Installation de Docker en local et mise en place d'un *Docker swarm* local
  - Machine virtuelle Ubuntu 18.04 sur serveur à la HES-SO Valais[4]
- Lancement des containers dans le code en utilisant le client Docker Java *spotify/docker-client* pour l'accès à l'API Docker
- Blocage du trafic réseau externe en permettant uniquement la communication interne en lien avec le cœur de *Docker swarm* (dans le code)
- Définition de quotas de RAM par container (dans le code)
- Définition de quotas de CPU par container (dans le code)
- Mettre les containers dans une queue et les exécuter seulement quand il y a assez de ressources disponibles
  - Utilisation et démonstration du mécanisme *scheduler/queue* dans *Docker swarm* (dans le code)

[1] Loi Fédérale sur la Protection des Données (Suisse)
[2] Révision de la LPD
[3] Réglement Général pour la Protection des Données (Europe)
[4] Haute École Spécialisée de Suisse Occidentale

- Interface web basique pour démonstration des fonctionnalités implémentées dans ce TB[5]

II. « Nice to have » (secondaires)

- Possibilité de fixer la durée de vie maximale d'un container (dans code)
- Analyse des alternatives à Docker Hub
- Mise en place d'un registre Docker (pour images) privé interne, avec intégration de l'authentification *Keycloak* à travers *OpenID Connect*
- Mise en place d'une procédure de « Conditions d'utilisation » pour les données
  - Pour les chercheurs (confidentialité des données et respect de règles de traitement)
  - Pour les patients (accord pour l'utilisation de leurs données personnelles / sensibles)

Des points peuvent être ajoutés ou modifiés dans le *Product Backlog* du projet, notamment en fonction des découvertes lors de l'état de l'art autour de Docker et l'analyse des risques.

Délais et livrables

I. Rendu et présentation

La documentation et autres fichiers liés au projet doivent être rendus le 3 décembre 2018 à 12h00. La date de défense orale sera définie plus tard.

[5] Travail de Bachelor

II. Livrables

- Rapport (avec annexes)
- Guide d'installation et utilisation
- Guide technique
- Présentation (Powerpoint ou Prezi)
- Code écrit

Lionel Engel (Étudiant) :

Date : ..... Signature : .....

Dr. Henning Müller (Professeur) :

Date : ..... Signature : .....

Ivan Eggel (Support technique) :

Date : ..... Signature : .....

Annexe II : Product backlog

US Nr.	Thème	*User Stories (US)*			Critères d'acceptation	Priorité	Statut	*Story Points*	Semaine	US acceptée	MoSCoW
		En tant que...	j'aimerais...	afin de...							
1	Preparation	Développeur	Preparer l'environment de travail et les modèles de documents	Etre régulier et à jour dans la rédaction	Documents créés et mis en forme + machine virtuelle créée par le Sinf	1000		1	0	10.10.2018	Must
2	Preparation	Responsable	Rédiger le cahier des charges avec le développeur	Définir les objectifs du projet avec le développeur	Cahier des charges complet avec des objectifs clairs et	990		2	0	10.10.2018	Must
3	Installation Docker	Développeur	Installer Docker en local	Pouvoir réaliser le développement sur Docker dans un environnement local séparé du projet global en cours	Docker installé selon les instructions d'installation et fonctionnel	980		1	1	17.10.2018	Must
4	Installation Docker	Développeur	Mettre en place un *Docker swarm* local	Pouvoir réaliser le développement sur Docker dans un environnement local séparé du projet global en cours	*Docker swarm* installé selon les instructions d'installation et fonctionnel	970		3	1	17.10.2018	Must
5	Analyse	Développeur	Faire une analyse des concepts et versions de Docker	Rédiger un état de l'art autour de Docker	Plusieurs sources bibliographiques analysées et rédaction dans le rapport	960		8	1	17.10.2018	Must
6	Analyse	Développeur	Faire une analyse des standards et mises en conformité de Docker	Rédiger un état de l'art autour de Docker	Plusieurs sources bibliographiques analysées et rédaction dans le rapport	950		5	1	24.10.2018	Must
7	Analyse	Développeur	Faire une analyse des outils et méthodes disponibles dans Docker	Rédiger un état de l'art autour de Docker	Plusieurs sources bibliographiques analysées et rédaction dans le rapport	940		8	2	24.10.2018	Must
11	Développement	Chercheur	Exécuter des containers dans mon code	Analyser des données et effectuer des opérations via mon code	Un container s'exécute avec succès via le client Docker Java	930		8	3	31.10.2018	Must
13	Développement	Chercheur	Définir des quotas de RAM par container	Éviter une surcharge des noeuds dans le cluster Docker swarm lors de l'exécution de containers	Le quota de RAM défini est pris en compte par le système	920		5	3	31.10.2018	Must
14	Développement	Chercheur	Définir des quotas de CPU par container	Éviter une surcharge des noeuds dans le cluster Docker swarm lors de l'exécution de containers	Le quota de CPU défini est pris en compte par le système	910		5	3	31.10.2018	Must
12	Développement	Chercheur	Bloquer le trafic réseau externe	Empêcher la communication d'informations sensibles vers l'extérieur	Aucun trafic ne sort du réseau interne défini	900		5	4	31.10.2018	Must
15	Développement	Chercheur	Mettre les containers dans une queue et les exécuter seulement quand il y a assez de ressources disponibles	Éviter une surcharge des noeuds dans le cluster Docker swarm lors de l'exécution de containers	Un container reste en attente tant qu'il n'y a pas de ressources disponibles	890		5	4	07.11.2018	Must

US Nr.	Thème	*User Stories (US)*			Critères d'acceptation	Priorité	Statut	*Story Points*	Semaine	US acceptée	MoSCoW
		En tant que...	j'aimerais...	afin de...							
16	Développement	Développeur	Mettre en place une interface web basique	Démontrer les fonctionnalités implémentées dans ce travail	Interface web de base créée et accessible via url	880		8	4		Must
8	Analyse	Développeur	Faire une analyse des risques du côté de l'infrastructure	Effectuer une analyse des risques d'exécution de code externe sur des machines internes	Risques relevés depuis plusieurs sources et rédaction dans le rapport	870		5	5	07.11.2018	Must
9	Analyse	Développeur	Faire une analyse des risques du côté de la sécurité	Effectuer une analyse des risques d'exécution de code externe sur des machines internes	Risques relevés depuis plusieurs sources et rédaction dans le rapport	860		8	6	27.11.2018	Must
10	Analyse	Développeur	Faire une analyse des risques du côté de la loi	Effectuer une analyse des risques d'exécution de code externe sur des machines internes	Risques relevés depuis plusieurs sources et rédaction dans le rapport	850		8	7	27.11.2018	Must
17	Développement	Chercheur	Pouvoir fixer la durée de vie maximale d'un container	Éviter une exécution en boucle qui ne s'arrête pas	L'exécution d'un container s'arrête lorsqu'il atteint la durée de vie définie	840		3			Should
18	Analyse	Responsable	Avoir une analyse des alternatives à Docker Hub	Pouvoir éventuellement être indépendant de Docker Hub	Alternatives listées, avec avantages, inconvénients, ressources nécessaires, temps estimé	830		8			Should
19	Développement	Chercheur	Mettre en place un registre Docker privé interne	Empêcher que mes codes soient accessibles publiquement	Des images peuvent être stockées, puis utilisées via cet emplacement interne	820		5	7	27.11.2018	Should
20	Développement	Chercheur	M'authentifier sur le registre Docker privé à travers OpenID Connect et Keycloak	Sécuriser mes accès en interne	L'accès est possible via ce système interne	810		8			Should
21	Développement	Chercheur	Lire et signer un accord de confidentialité des données	Respecter les règles (légales) de traitement liés aux données sensibles	Possibilité d'accéder et de signer l'accord	800		5			Should
22	Développement	Patient	Lire et signer un accord pour l'utilisation de mes données personnelles	Comprendre et valider les conditions de traitements de données et mes droits qui y sont liés	Possibilité d'accéder et de signer l'accord facilement et explicitement	790		5			Should
23	Analyse	Développeur	Analyser les différences de fonctionnement de Docker (développement effectué) sur Windows et sur Linux	Avoir une comparaison et pouvoir intégrer Windows dans le projet	Différences analysées de manière complète	780		5			Could
							TOTAL	124			

Annexe III : PV séance kick-off 02.10.2018

But de la séance	Séance kick-off TB
Date et heure	02 octobre 2018 10h00 – 11h00
Lieu	Technopôle (IIG)
Participants	Lionel Engel, Dr. Henning Müller, Ivan Eggel

Point	Par qui	Description
1		**Administration / organisation**
1.1	Henning et Lionel	Structure / contenu du rapport • Mettre en forme la structure dès le début • Rédiger un texte pour l'expert (« qui ne connaît pas ou partiellement ») • Pour les sources de type URL : note de bas de page avec la date indiquée (pas dans références/sources à la fin) • Donné un exemple de TB à Lionel • Lionel a indiqué vouloir faire son TB en français (information nécessaire pour le choix de l'expert)
1.2	Henning	Défense orale • Date indiquée plus tard • Contenu : revue de l'existant (ouvrages, …), choix effectués, démo, amélioration possible, conclusion
1.3	Henning	Séances • Hebdomadaires dans la mesure du possible • Parfois avec Henning, parfois Ivan, parfois les deux
1.4	Henning	Rôles • Henning : professeur qui suit le TB. Organisation, rédaction, cahier des charges • Ivan : cahier des charges, aspects techniques
1.5	Henning et Lionel	• Signature de la feuille de données du TB • Lionel va apporter aujourd'hui la feuille à l'assistante en charge du processus des TB
2		**Présentation cahier des charges et de l'existant**
2.1	Ivan	Cahier des charges • Présentation de ce qui a été déjà rédigé (proposition) • Lionel doit reprendre ce cahier des charges, le développer et effectuer des propositions (présentation au plus tard à la prochaine séance)
2.2	Ivan	Projet existant • Présentation du projet (Fonctionnement, Docker, moyens utilisés) • Démo (Docker, interface Web) • Envoi de documentation à Lionel par email (Présentation pptx, rapport scientifique pour ce projet)

Point	Par qui	Description
2.3	Lionel	Lionel doit demander une machine virtuelle linux au Service informatique de la HES-SO Valais pour la réalisation du TB
3		**Prochaine séance**
3.1	Tous	10.10.2018 13h00 Technopôle

Annexe IV : PV séance 2 10.10.2018

But de la séance	Séance définition / validation cahier des charges et product backlog
Date et heure	10 octobre 2018 13h00 – 14h00
Lieu	Technopôle (IIG)
Participants	Lionel Engel, Dr. Henning Müller, Ivan Eggel

Point	Par qui	Description
1		**Avancement du travail**
1.1	Lionel	Ce que j'ai fait : • Mise en place structure des documents • Lecture documents envoyés par Ivan sur ce qui a été fait dans le projet global • Recherches sur Docker • Analyse et complétion du cahier des charges proposé • Rédaction des User stories dans le Product Backlog
1.2	Lionel	Ce que je vais faire : • Compléter le product backlog (story points, tâches) • Amener cahier des charges validé et product backlog à l'administration • Installer Docker et *Docker swarm* en local • Recherches sur Docker (cf. product backlog) et rédaction de l'état de l'art
1.3	Lionel	Problèmes rencontrés : • Opération cardiaque d'un membre de la famille => m'en occuper • Doutes concernant la réalisation du product backlog => j'ai eu des réponses de Henning Müller
2		**Cahier des charges / Product Backlog**
2.1	Tous	Validation du cahier des charges
2.2	Tous	Révision et validation des user stories • Mettre « implémentation d'interface Web basique » en priorité la plus faible des « must » (Juste pour démo lors de la défense) • Importance d'installer Docker avant de faire les recherches (pour faire des tests, meilleure analyse)
3		**Aspects techniques**
3.1	Ivan et Lionel	Lionel voudrait à priori utiliser la technologie NodeJS pour réaliser l'interface Web basique de démonstration => Remarque d'Ivan : il faudra mettre en place un Web service pour le lien avec le client Java pour Docker
4		**Prochaine séance**
4.1	Ivan et Lionel	Mercredi 17.10.2018 à 13h00

Annexe V : PV séance 3 17.10.2018

But de la séance	Séance de suivi du TB
Date et heure	17 octobre 2018 13h00 – 13h15
Lieu	Technopôle (IIG)
Participants	Lionel Engel, Ivan Eggel

Point	Par qui	Description
1		**Avancement du travail**
1.1	Lionel	Ce que j'ai fait : • Installé Docker en local sur VM Ubuntu • Mis en place Docker swarm local • Installé une deuxième VM sur mon ordinateur pour tester le Docker swarm • Commencé à analyser les principes (concepts) et les versions de Docker
1.2	Lionel	Ce que je vais faire : • Compléter l'analyse des principes et versions de Docker • Analyser les standards de Docker • Analyser les outils et méthodes de Docker • Effectuer l'analyse des risques du côté de l'infrastructure
1.3	Lionel	Problèmes rencontrés : • Pas de problèmes en particulier
2		**Aspects techniques**
2.1	Ivan	Ivan dit que c'est très bien d'utiliser les fichiers Docker-compose pour la configuration des services Docker dans la mesure du possible (quotas, …)
2.2	Ivan	Ce serait bien de rajouter dans les « Nice to have » une comparaison du fonctionnement entre Docker sur Windows et sur Linux => Lionel l'ajoute à la fin du Product backlog
3		**Prochaine séance**
3.1	Henning, Ivan et Lionel	Mercredi 24.10.2018 à 13h30

Annexe VI : PV séance 4 24.10.2018

But de la séance	Séance de suivi du TB
Date et heure	24 octobre 2018 13h30 – 14h00
Lieu	Technopôle (IIG)
Participants	Lionel Engel, Henning Müller, Ivan Eggel

Point	Par qui	Description
1		**Avancement du travail**
1.1	Lionel	Ce que j'ai fait : • Finir analyse des principes et versions de Docker • Analyser les standards / best practices Docker • Analyser les outils et méthodes de Docker • Commencer l'analyse des risques pour l'infrastructure
1.2	Lionel	Ce que je vais faire : • Exécution des containers dans le code • Quotas RAM et CPU (tests avec docker run, puis intégrer dans le code) • Analyse de risques liés à l'infrastructure (si temps)
1.3	Lionel	Problèmes rencontrés : • Difficultés à trouver d'autres risques que ceux que j'ai pu déjà constater dans en lisant des références bibliographiques
3		**Rapport**
3.1	Henning	Henning Müller m'a déjà envoyé par email des commentaires par rapport au document => lui renvoyer plus tard avec les ajouts / modifications apportés. « Structure plutôt bien »
3.2	Ivan	Conseillé de fusionner si possible les petites sections de 1-2 phrases
4		**Modifications Product backlog**
4.1	Ivan et Lionel	Mettre les US « containers dans le code », « quotas CPU » et « quotas RAM » en priorité plus élevées que l'analyse des risques (pratique = meilleures compréhension et pour pouvoir avancer sur le développement)
5		**Prochaine séance**
5.1	Henning, Ivan et Lionel	A définir

Annexe VII : PV séance 5 31.10.2018

But de la séance	Séance de suivi du TB
Date et heure	31 octobre 2018 13h30 – 14h00
Lieu	Technopôle (IIG)
Participants	Lionel Engel, Henning Müller, Ivan Eggel

Point	Par qui	Description
1		**Avancement du travail**

1.1	Lionel	Ce que j'ai fait :
		• Exécution des containers dans le code
		  ○ Spring boot application qui utilise spotify/docker-client
		• Quotas RAM et CPU
		  ○ CPU seulement défini au niveau de la tâche (nanoCPU) => à compléter / vérifier

1.2	Lionel	Ce que je vais faire :
		• Compléter / vérifier quotas CPU
		  ○ Fonctionnement nanoCPU
		  ○ Que se passe-t-il si on dépasse ? (aussi quota RAM)

1.3	Lionel	Problèmes rencontrés :
		• Problème lié aux dépendences Maven (pom.xml) => erreur (cause inconnue)
		  ○ Résolu en ajoutant une dépendence (avec aide d'Ivan)
		• Service Docker créé dans le web service Spring => les containers redémarrent en boucle à cause de la limite mémoire trop basse
		  ○ Résolu en ajoutant un maximum de ré-exécutions de containers

2		**Rapport**
2.1	Lionel	Section « développement » => Contenu ? (clarifications) Réponse : Mon développement avec tests et résultats (fonctionne, quels sont les problèmes, overhead, …)
3		**Prochaine séance**
4.1	Henning, Ivan et Lionel	Mercredi 07.11.18 09h00

Annexe VIII : PV séance 6 06.11.2018

But de la séance	Séance de suivi du TB
Date et heure	06 novembre 2018 09h00 – 09h30
Lieu	Technopôle (IIG)
Participants	Lionel Engel, Henning Müller, Ivan Eggel

Point	Par qui	Description
1		**Avancement du travail**
1.1	Lionel	Ce que j'ai fait : • Vérifier fonctionnement quotas CPU • Bloquer le trafic réseau externe • Mettre en place une interface Web basique
1.2	Lionel	Ce que je vais faire : • Principe de scheduler / queue • Rédiger dans le rapport tout ce que j'ai écrit « à la main » lors de mes découvertes / du développement
1.3	Lionel	Problèmes rencontrés : • Impossible d'accéder au port dédié à mon Web service depuis un réseau externe, ainsi que via le VPN => à résoudre avec le Service informatique
2		**Rapport**
2.1	Lionel	A compléter et envoyer à Henning d'ici vendredi maximum pour relecture
3		**Prochaine séance**
3.1	Henning, Ivan et Lionel	Mardi 13 novembre à 9h00

Annexe IX : PV séance 7 13.11.2018

But de la séance	Séance de suivi du TB
Date et heure	13 novembre 2018 09h00 – 09h30
Lieu	Technopôle (IIG)
Participants	Lionel Engel, Henning Müller, Ivan Eggel

Point	Par qui	Description
1		**Avancement du travail**
1.1	Lionel	Ce que j'ai fait : • Rédiger dans le rapport tout ce que j'ai écrit « à la main » lors de mes découvertes / du développement • Compléter les quotas de RAM dans le rapport • Principe de scheduler / queue => recherches sur le fonctionnement et rédaction
1.2	Lionel	Ce que je vais faire : • Compléter l'analyse des quotas CPU de manière plus approfondie (CFS scheduler et real-time scheduler) • Compléter la partie sur l'isolation réseau des conteneurs en fonction de la discussion avec Henning et Ivan • Analyse des risques
1.3	Lionel	Problèmes rencontrés : • Impossible d'accéder à la machine virtuelle un jour • Pas assez rédigé mes tests et résultats dans le rapport => complété ce weekend • Besoin de précisions concernant les réseaux de conteneurs
2		**Rapport**
2.1	Henning, Ivan et Lionel	Lionel a envoyé une version vendredi 9 novembre à Henning et Ivan Corrections effectuées en fonction de leur feedback reçu entre vendredi et dimanche par email La partie « Résultats » doit « être le 50 % au moins du rapport »
		Ajouter scénarios éventuels d'application pour le Swarm scheduler
3		**Isolation réseau des conteneurs**
3.1	Ivan et Henning	Envoyé des compléments d'information à Lionel • Lionel doit tester l'accès au NAS (*mount* de l'hôte), avec réseau isolé • Le conteneur doit être complétement isolé
4		**Prochaine séance**
4.1	Henning, Ivan et Lionel	Mercredi 21.11.2018 à 13h30

Annexe X : PV séance 8 21.11.2018

But de la séance	Séance de suivi du TB
Date et heure	21 novembre 2018 13h30 – 14h30
Lieu	Technopôle (IIG)
Participants	Lionel Engel, Henning Müller, Ivan Eggel

Point	Par qui	Description
1		**Avancement du travail**
1.1	Lionel	Ce que j'ai fait : • Compléter l'analyse des quotas CPU de manière plus approfondie (CFS scheduler et realtime scheduler) • Compléter la partie sur l'isolation réseau des conteneurs en fonction de la discussion avec Henning et Ivan • Compléter les résultats dans le rapport en fonction de commentaires reçus par email • Analyse des risques (~70%)
1.2	Lionel	Ce que je vais faire : • Finir l'analyse des risques • Compléter la section Résultats en fonction de la discussion avec Henning et Ivan
1.3	Lionel	Problèmes rencontrés : • Il n'y a apparemment pas assez d'éléments dans la section Résultats => à discuter avec Henning et Ivan
2		**Administration**
2.1	Henning, Lionel	Il faudrait que Henning envoie son adresse Urkund pour dépôt du TB le 3.12 ⇨ Il va regarder ça
3		**Rapport**
3.1	Lionel, Ivan et Henning	Que manque-t-il dans les résultats ? Envoi du rapport à Ivan et Henning et ils me feront les commentaires
3.2	Lionel, Ivan et Henning	Analyse des risques • Il est important de pouvoir identifier les limites, ne pas dire « mon logiciel est infaillible à 100%, parfait ». Mais pas besoin d'écrire 20 pages sur ça • Mentionner (et tester si temps disponible) des solutions pour combler ces risques
3.2	Lionel	Faut-il faire un user guide et tech guide ? Pas besoin user guide, mais tech guide oui (pour Ivan)
4		**Objectifs « nice to have » / continuation après 3 décembre**
4.1	Lionel	Lionel va continuer à travailler sur des éléments manquants après dépôt du rapport
5		**Prochaine (et dernière) séance**
5.1	Henning, Ivan et Lionel	

Annexe XI : Liste de commandes Dockerfile[51]

Commande	Description	Syntaxe	Best practice[52]
FROM	Définition de l'image de base à utiliser	FROM <nom image> : [<version>] *ex : FROM ubuntu : 16.04*	✓ Privilégier l'utilisation des dépôts d'images officiels ✓ Utiliser des images adaptées aux besoins (taille, …)
MAINTAINER	Informations sur l'auteur de l'image	MAINTAINER <infos auteur> *ex : MAINTAINER J. Doe <j.doe @gmail.com>*	
COPY	Copie de fichiers depuis le système hôte Docker vers le système de fichiers dans l'image générée	COPY <sources séparées par un espace> <destination> *ex : COPY apache2.conf example /etc/apache2/conf/*	✓ Si nous devons copier plusieurs fichiers, il faut les copier de manière individuelle pour éviter des invalidations de caches

Tableau 3 Liste instructions Dockerfile

[51] Descriptions et syntaxes récupérées et traduites en français depuis l'ouvrage de J. Chelladhurai *et al.* (2017)

[52] Récupérées et traduites en français depuis https://docs.docker.com/develop/develop-images/dockerfile_best-practices/#dockerfile-instructions, le 24 octobre 2018

Commande	Description	Syntaxe	Best practice[53]
ADD	Idem que COPY, mais prend aussi en charge les fichiers compressés *.tar et les adresses URL distantes	ADD <sources séparées par un espace> <destination> *ex : ADD example.tar /var/example/*	✓ Utiliser seulement pour les dossiers compressés (.tar). Pour les URL distantes, utiliser plutôt *curl* ou *wget* *RUN mkdir -p /var/tmp/example \ && wget http://example.com/file.tar.xz \ | tar -xJC /var/tmp/example*
ENV	Définition d'une variable d'environnement	ENV <clé> <valeur *ex : ENV APACHE_LOG_DIR /var/log/apache/*	
USER	Indication de l'utilisateur par défaut (Si non spécifié, Docker va utiliser *root*)	USER <UID> ou <nom utilisateur> *ex : USER john*	✓ Éviter d'utiliser la commande *sudo* (comportement instable) pour élever les privilèges utilisateurs ✓ Éviter les utilisations de l'instruction USER à répétition dans un même fichier

Tableau 3 Liste instructions Dockerfile (suite)

[53] Récupérées et traduites en français depuis https://docs.docker.com/develop/develop-images/dockerfile_best-practices/#dockerfile-instructions, le 24 octobre 2018

Commande	Description	Syntaxe	Best practice[54]
WORKDIR	Définition de l'emplacement de travail	WORKDIR <chemin de dossier> *ex : WORKDIR /var/html/*	✓ Utiliser des chemins absolus pour les répertoires *WORKDIR /var/www/dossier1/* et non *WORKDIR ./dossier1* ✓ Utiliser cette instruction à la place de *RUN cd <nom répertoire>*
VOLUME	Définition d'un volume qui peut être utilisé comme point de montage depuis l'hôte ou d'autres conteneurs.	VOLUME <chemin de montage> *ex : VOLUME /tmp/monVolume/*	
EXPOSE	Ouverture d'un port réseau du conteneur vers l'extérieur	EXPOSE <port1> [/<protocole1>] [<port2> </protocole2>] … *ex : EXPOSE 4000/udp 80*	✓ Utiliser en priorité les ports réseau standards (80 pour http, 21 pour ftp, …)

Tableau 3 Liste instructions Dockerfile (suite)

[54] Récupérées et traduites en français depuis https://docs.docker.com/develop/develop-images/dockerfile_best-practices/#dockerfile-instructions, le 24 octobre 2018

Commande	Description	Syntaxe	Best practice[55]
LABEL	Spécification de données utilisées pour la gestion / orchestration de l'image	LABEL <clé1>=<valeur1> <clé2>=<valeur2> … *ex : LABEL version="1.0" creation-date="2018-10-01"*	
RUN	Exécution de commandes (il est recommandé d'exécuter plusieurs commandes, séparées par '&&', dans une seule instruction RUN pour éviter des *layers* supplémentaires)	RUN <commande> *ex : RUN apt-get update && apt-get install -y mysql-server* Ou RUN ["<exécutable>", "<argu1>", "<argu2>"] *ex : RUN ["bash", "-c", "mkdir", "/tmp/dossier"]*	✓ Séparer les longues instructions sur plusieurs lignes à l'aide du '\' (meilleure lisibilité et compréhension) ✓ Éviter d'utiliser *apt-get upgrade* ou *apt-get dist-upgrade* (à cause des droits d'accès et exécution notamment) ✓ Utiliser *apt-get update* et *apt-get install* dans la même instruction pour éviter l'utilisation d'une version *update* mise en cache Docker (en cas d'une précédente création d'image)

Tableau 3 Liste instructions Dockerfile (suite)

[55] Récupérées et traduites en français depuis https://docs.docker.com/develop/develop-images/dockerfile_best-practices/#dockerfile-instructions, le 24 octobre 2018

Commande	Description	Syntaxe	Best practice[56]
CMD	Idem que RUN, également pour les applications. Cependant, CMD sera exécutée au moment du lancement du conteneur et non pas lors de la création comme RUN	CMD <commande> *ex : CMD ping 8.8.8.8* Ou CMD [“<exécutable>”, “<argu1>”, “<argu2>”] *ex : CMD [“ping”, “8.8.8.8”]*	✓ Privilégier la forme *CMD [“executable”, “param1”, “param2” …]*
ENTRYPOINT	Exécution d’une application durant toute la durée de vie du conteneur. Le conteneur agit en quelque sorte comme un exécutable.	ENTRYPOINT <commande> *ex : ENTRYPOINT echo “Hello world”* Ou ENTRYPOINT [“<exécutable>”, “<argu1>”, “<argu2>”] *ex : ENTRYPOINT [“echo”, “Hello world”]*	

Tableau 3 Liste instructions Dockerfile (suite)

[56] Récupérées et traduites en français depuis https://docs.docker.com/develop/develop-images/dockerfile_best-practices/#dockerfile-instructions, le 24 octobre 2018

Commande	Description	Syntaxe	Best practice[57]
HEALTHCHECK	Vérification de la santé d'une application au sein d'un conteneur (et donc du conteneur en lui-même). Il est possible de modifier la fréquence d'appel de l'instruction (*--interval*), le délai d'expiration (*--timeout*) et le nombre de nouvelle tentative avant que le conteneur soit « non sain » (*--retries*)	HEALTHCHECK [<options>] CMD <commande> *ex: HEALTHCHECK --interval=3m --timeout=20s --retries=2 CMD ping 8.8.8.8*	
ONBUILD	Définition d'une instruction qui est exécutée lorsqu'une autre image est créée à partir de cette image de base. Il n'est pas possible d'entrer un FROM ou MAINTAINER en tant qu'instruction ONBUILD	ONBUILD <instruction> *ex: ONBUILD RUN apt-get update*	Faire attention lors de l'utilisation de ADD ou COPY. ONBUILD risque d'échouer sur l'image « enfant » si les ressources mentionnées sont manquantes dans le contexte de la nouvelle *build*.

Tableau 3 Liste instructions Dockerfile (suite)

[57] Récupérées et traduites en français depuis https://docs.docker.com/develop/develop-images/dockerfile_best-practices/#dockerfile-instructions, le 24 octobre 2018

Commande	Description	Syntaxe	Best practice[58]
STOPSIGNAL	Définition d'un signal d'arrêt (sortie) du conteneur	STOPSIGNAL <signal> *ex: STOPSIGNAL SIGKILL*	
SHELL	Spécification d'un autre *shell* pour remplacer *sh* Linux ou *cmd* Windows	SHELL ["<shell>", "<argu1>", "<argu2>", …] *ex: SHELL ["powershell", "-command"]*	

Tableau 3 Liste instructions Dockerfile (suite)

[58] Récupérées et traduites en français depuis https://docs.docker.com/develop/develop-images/dockerfile_best-practices/#dockerfile-instructions, le 24 octobre 2018

Déclaration sur l'honneur

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 Responsable de filière 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 :

- Dr. Henning Müller
- M. Ivan Eggel

Sierre, le 30 novembre 2018

Lionel Engel