ÉCOLE DE TECHNOLOGIE SUPÉRIEURE
UNIVERSITÉ DU QUÉBEC

RAPPORT DE MAÎTRISE PRÉSENTÉ À
L'ÉCOLE DE TECHNOLOGIE SUPÉRIEURE

COMME EXIGENCE PARTIELLE
À L'OBTENTION DE LA
MAÎTRISE EN GÉNIE DES TECHNOLOGIES DE L'INFORMATION
M. ING.

PAR
EVINS DULIENCE

DOCKER : POUR UNE UTILISATION OPTIMALE DES ESPACES DE NOMS ET DES GROUPES DE CONTRÔLE

MONTRÉAL, LE 7 OCTOBRE 2015

PRÉSENTATION DU JURY

CE MÉMOIRE A ÉTÉ ÉVALUÉ

PAR UN JURY COMPOSÉ DE :

M. Alain April, directeur de mémoire
Département de Génie Logiciel à l'École de technologie supérieure

M. Abdelaoued Gherbi, membre du jury
Département de Génie Logiciel à l'École de technologie supérieure

DOCKER: POUR UNE UTILISATION OPTIMALE DES ESPACES DE NOMS ET DES GROUPES DE CONTRÔLE

EVINS DULIENCE

RÉSUMÉ

L'objectif principal de cette recherche est d'expérimenter et d'évaluer les procédés fournis par Docker pour gérer les ressources du serveur hôte. Docker est une technologie à base de conteneur et comme pour toute autre technologie à base de conteneur, Docker se sert du même noyau et des mêmes librairies du système d'exploitation hôte pour exécuter ses conteneurs. Pour l'instant, Docker est compatible seulement au système d'exploitation GNU Linux. Pour être une technologie de virtualisation nouvelle à base de conteneur, il a été primordial de le situer un peu par rapport à la virtualisation de type hyperviseur telle que VMWare ou Oracle VirtualBox. Ces technologies, différemment à Docker, font une répartition des ressources physiques du serveur hôte afin que chaque machine virtuelle puisse exécuter leur propre système d'exploitation.

Pour gérer les ressources du serveur hôte et assurer l'isolation de tous ses conteneurs, Docker utilise en arrière-plan les fonctionnalités «Espace de noms» et «Groupe de Contrôle» du système d'exploitation Linux. Les conteneurs Docker s'exécutent dans des espaces de noms Linux créés en même temps que les conteneurs et qui peuvent être directement manipulés à l'aide des outils appropriés du SE tel que systemctl. Les ressources mémoire, processeurs et disques dur sont celles sur lesquelles les expérimentations ont été effectuées. Les résultats prouvent que Docker prend très bien en charge la gestion de la mémoire et du processeur, quoiqu'il reste des améliorations à faire. Le disque dur (c.-à-d. bande passante disque et espace de stockage) pour lequel des solutions de gestion de ressources ont été fournies et le réseau (c.-à-d. bande passante) pour lequel rien n'a été proposé pour l'instant constituent les aspects à regarder dans les prochaines versions en vue d'avoir une plateforme Docker plus stable méritant la pleine confiance des professionnels TI.

Mot clés : docker, virtualisation à base de conteneur, espace de noms, groupe de contrôle, gestion de ressources

DOCKER: FOR AN OPTIMAL USE OF NAMESPACES AND CONTROL GROUPS

EVINS DULIENCE

ABSTRACT

The main objective of this research is to test and evaluate the means provided by Docker to manage the resources of the host server. Docker is a container-based technology and as any other container-based technology, Docker uses the same kernel and libraries of the host operating system to run its containers. Currently, Docker is only compatible with GNU Linux operating systems. As a new container-based virtualization technology, it is important to determine how it relates to the hypervisor-based virtualization technology, like VMware or Oracle VirtualBox. These implementations, unlike Docker, partition the physical resources of the host server so that each virtual machine can run their own operating system.

To manage the resources of the host server and provide isolation of all its containers, Docker uses the features "Namespaces" and "Control Groups" of the Linux Operating System. Docker containers run in Linux namespaces created together with the containers that can be directly manipulated using OS appropriate tools such as systemctl.

Memory resources, CPU and hard disk are those on which the experiments were conducted. The results show that Docker can handle memory and processor management very well, though there is still room for improvement. In order to have a more stable Docker platform that deserves the full confidence of the IT professionals, the hard disk drive (i.e. disk bandwidth and disk storage space) and the network (i.e. network bandwidth) are the aspects to look at in future versions. Presently, a few options are provided to manage hard drive resources however nothing is provided as yet for the network bandwidth management.

Key words: docker, container-based virtualization, namespace, control groups, resource management

TABLES DES MATIÈRES

Page

LISTE DES FIGURES

Page

LISTE DES ABRÉVIATIONS, SIGLES ET ACRONYMES

TI	Technologie de l'Information
VM	Virtual Machine
SE	Système d'Exploitation
VMM	Virtual Machine Monitor
PaaS	Platform as a Service
IMM	Integrated Modular Avionics
PID	Process Identifier
Cgroups	Control Groups
tc	Traffic Controller
RAM	Random Access Memory
FSS	Fair Share Scheduling
CFS	Completely Fair Scheduler
IOPs	Input/Output Operations Per Second
MB/s	Megabyte Per Second

INTRODUCTION

Depuis environ une dizaine d'années, la virtualisation se révèle presque incontournable dans le domaine des technologies de l'information (TI) et est redevenue récemment un sujet d'actualité (Wang et *coll*. 2010) et (Xavier et *coll*. 2013). Le domaine de la virtualisation est en pleine croissance et attire la convoitise de nombreux grands acteurs informatiques qui travaillent d'arrachepied pour se créer une place et prendre ce marché prometteur (Asay, 2013). « La virtualisation est une technologie logicielle éprouvée offrant la possibilité d'exécuter simultanément plusieurs systèmes d'exploitation et applications sur un même serveur. Celle-ci bouleverse rapidement le paysage informatique en modifiant fondamentalement nos usages de la technologie » (VMware, 2015). La virtualisation devient aussi, de jour en jour, une facette importante de l'architecture technique des entreprises qui cherchent, à tout prix, à optimiser leur budget d'équipements informatiques et aussi assurer une gestion efficace de l'espace physique de leur centre de données.

Traditionnellement, la tendance était d'utiliser un serveur physique pour chaque application ou peut-être plusieurs applications compatibles pouvaient cohabiter sur un même serveur physique. Au fil du temps, les entreprises constatent qu'elles dépensent beaucoup trop de budgets pour des serveurs souvent sous-utilisés, sans compter leurs couts de maintenance. Donc, il fallait trouver un moyen permettant d'optimiser cette situation. D'où l'adoption rapide de la technologie de virtualisation qui, aujourd'hui, est considérée comme un outil primordial pour les professionnels d'infrastructure des TI, car elle est à la base de bien d'autres technologies (ISACA, 2010), telle que : l'informatique en nuage et l'avionique modulaire intégrée (IMA) (Han et Jin, 2011). Avec cette nouvelle technologie, désormais, il n'est plus nécessaire de dédier un serveur physique pour chaque application, et il est possible de consolider plusieurs applications sur un seul serveur physique, chacune dans un environnement isolé et partageant les ressources disponibles.

Maintenant qu'il est possible d'exécuter plusieurs applications sur un seul serveur physique, sans à se soucier de leur compatibilité, car chaque machine virtuelle (VM) est isolée l'une de l'autre, il devient important de bien gérer les ressources du serveur puisque les applications,

qui cohabitent sur le serveur, sont en compétitions pour l'utilisation des mêmes ressources. Si pour une raison quelconque, l'une des VM commence à consommer les ressources de manière inappropriée, cela va sans doute avoir des conséquences sur les autres VM qui cohabitent sur ce serveur. Conséquemment, il faut s'assurer de l'isolation de la performance de chaque VM. Il est aussi nécessaire d'allouer des ressources suffisantes et nécessaires à chaque VM de manière à s'assurer qu'elles peuvent effectuer correctement leurs tâches sans causer de préjudice au serveur physique et aux autres applications.

Docker est un nouveau type de technologie de virtualisation dont sa première version a été lancée le 20 mars 2013 (Docker, 2014). « C'est une plateforme ouverte pour développer, expédier et exécuter des applications. Docker est conçu pour délivrer les applications plus rapidement. Avec Docker vous pouvez séparer vos applications de votre infrastructure et de traiter votre infrastructure comme une application gérée … » (Docker, 2015)

Ce projet de recherche, de 15 crédits, à la maitrise en technologies de l'information, a pour objectif principal d'explorer et d'évaluer la gestion des ressources des conteneurs Docker en utilisant les lignes de commande proposées et les fonctionnalités « espaces de noms » et « groupes de contrôle » fournies par le noyau du système d'exploitation Linux, afin de proposer de bonnes pratiques à mettre en place en vue d'exécuter Docker dans un environnement de production.

Pour être une technologie en pleine croissance et n'ayant pas encore atteint la confiance de tous les professionnels TI pour être implémenté officiellement dans un environnement de production, il est encore important de bien évaluer les options fournies par Docker pour la gestion des ressources ; et aussi d'examiner le comportement des conteneurs et du serveur hôte dans chaque situation d'allocation de ressources. D'où l'objectif principal de ce travail de recherche. Ces options sont utilisées pour gérer la mémoire (c.-à-d. RAM, SWAP, etc …), le temps processeur (c.-à-d. processeur dédié, partage de temps processeur, etc.), les entrées-sorties disque et la bande passante réseau des conteneurs Docker créés au cours de cette recherche. Pour ce faire, un environnement de laboratoire est implémenté pour essayer et

expérimenter les différentes options décrites dans la documentation de Docker. Ces expérimentations permettent de jauger la stabilité de la technologie en termes de gestion de ressources qui est un aspect prépondérant pour le domaine de la virtualisation. Par la suite, le résultat obtenu pour chaque essai est présenté (voir le chapitre 3).

Docker, étant une technologie qui exploite principalement la puissance des fonctionnalités «Espace de noms » et « Groupes de contrôle » du noyau Linux, notamment pour fournir l'isolation de ses conteneurs et aussi gérer les ressources, il a été défini comme un objectif secondaire d'examiner un peu au cours de cette recherche, la manière dont Docker se sert de ces deux fonctionnalités.

CHAPITRE 1

Revue de la littérature

Cette revue littéraire a pour objectif de faire une introduction sommaire des différents thèmes utilisés dans ce projet et aussi de situer la virtualisation à base de conteneur par rapport aux autres types de virtualisation. Afin de présenter l'état de l'art de ce domaine, il a été nécessaire d'examiner différentes publications et revues spécialisées traitant de ce thème et suivre les progrès du domaine, sur le web et dans des groupes d'intérêts spécialisés, tout au long du projet.

Ce chapitre débute par la présentation de deux grands types de virtualisation. Par la suite, une discussion concernant la nécessité de gérer les ressources des serveurs physiques dans un environnement virtualisé est présentée. Finalement, il se termine par une synthèse des notions utiles pour cette recherche.

La virtualisation, depuis ses tout débuts, a été perçue comme un moyen d'optimiser les ressources informatiques des entreprises et, du même coup, permettre d'optimiser ses couts. Les recherches effectuées dans ce domaine ont permis l'émergence de plusieurs types de virtualisation qui comportent chacune leurs avantages et inconvénients.

Pour bien situer ce projet de recherche de 15 crédits, à la maitrise en technologie de l'information, il parait nécessaire de faire une brève introduction, tout d'abord, de deux grands types de solutions de virtualisation: 1) La virtualisation basée sur la technologie de type hyperviseur ; et 2) la virtualisation à base de conteneur. Docker (Docker, 2015) se présente comme l'une des technologies, à base de conteneur, les plus prometteuses au cours de ces deux dernières années (Konrad, 2015).

1.1 La virtualisation basée sur la technologie de type hyperviseur

Cette approche consiste à utiliser un logiciel intermédiaire (c.-à-d. un hyperviseur) qui a le rôle de présenter les composants physiques du serveur hôte en composants virtuels au serveur invité que l'on appelle communément la machine virtuelle (VM). De cette manière, la VM pourra interagir directement avec ses composants virtuels comme s'il s'agissait d'un serveur physique. L'hyperviseur est aussi appelé VMM (Virtual Machine Monitor), car il permet de contrôler les ressources allouées aux serveurs invités en plus de permettre une gestion centralisée de ceux-ci.

L'avantage, avec cette approche, est que l'on peut exécuter plusieurs systèmes d'exploitation de manière complètement indépendante sur une seule hôte. Chaque VM retrouvera tous les éléments nécessaires à l'installation de son propre système d'exploitation (SE). Cette approche permet aussi d'exécuter des systèmes d'exploitation de différentes distributions, par exemple, exécuter des VM Ubuntu, Solaris et CentOS concurremment à un système d'exploitation Windows. On peut y installer n'importe SE pourvu qu'il soit compatible avec l'architecture du serveur physique. Parmi les grands acteurs de virtualisation de type hyperviseur, on peut citer : VMware (VMware Inc., 2015), Xen (The Linux Foundation, 2013), Microsoft HyperV (Microsoft Corporation, 2015) ou VirtualBox (Oracle Corporation, 2015).

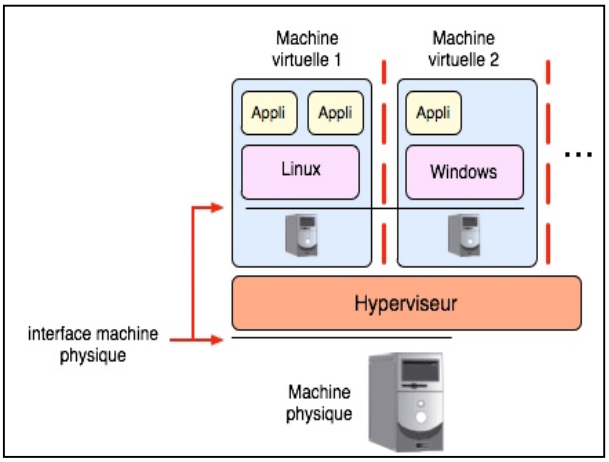

Figure 1.1. Hyperviseur et machine virtuelle (Krakowiak, 2014)

1.2 La virtualisation à base de conteneur

Différente du type de virtualisation présentée précédemment, la virtualisation à base de conteneur (aussi nommée virtualisation au niveau du système d'exploitation ou encore « Lightweight Virtualization ») ne virtualise pas le serveur physique, mais plutôt l'environnement d'exécution. Cette approche partitionne les ressources du serveur en créant des espaces isolés sur le même système d'exploitation. Les conteneurs vont s'exécuter à l'intérieur de ces différentes instances d'espace isolé. Chaque conteneur utilise le même noyau du système d'exploitation hôte et il a l'impression qu'il a son propre espace disque, mémoire, processeur, processus, périphériques, etc. Étant donné que ces conteneurs s'exécutent sur la même instance de système d'exploitation et partage le même noyau, tous les processus peuvent être vus à partir cette dernière; cependant une fois à l'intérieur du conteneur, les ressources sont complètement isolées par rapport au serveur hôte. Dans un environnement utilisant le système d'exploitation Linux, ce niveau d'isolation est rendu possible grâce aux fonctionnalités « groupes de contrôle » et « espaces de noms » du noyau du système d'exploitation. Parmi les grands acteurs de virtualisation de ce type on peut citer : LXC (LinuxContainer.org, 2015), Solaris zone (Oracle Corporation, 2015) et Docker (Docker inc., 2015).

Le tableau 1.1 présente une brève comparaison entre les différentes caractéristiques de la machine virtuelle et du conteneur.

Tableau 1.1. VM vs Conteneur (Dua, 2014)

Paramètres	**Machines Virtuelles**	**Conteneurs**
OS invité	Chaque VM s'exécute sur des périphériques virtuels et le noyau est chargé dans sa propre zone mémoire	Tous les systèmes invités utilisent le même système d'exploitation et le même noyau. L'image du noyau est chargée dans la mémoire physique.
Communication	Se fait via les périphériques Ethernet	Utilise des procédures IPC standard comme signal, pipe, soquet, etc.
Sécurité	Dépends de l'implémentation de l'Hyperviseur	Différents contrôles d'accès peuvent être exploités

Performance	Les machines virtuelles souffrent d'un petit manque de performance dû au fait que les instructions-machine doivent être traduites du système invité au système hôte.	En comparaison à leurs systèmes hôtes, les conteneurs expérimentent une performance quasi native.
Isolation	Le partage de librairies, fichiers, etc. entre le système d'exploitation invité et le système d'exploitation hôte n'est pas permis.	Les sous-répertoires peuvent être montés et partagés de façon transparente.
Temps de démarrage	Les VMs prennent quelques minutes pour démarrer	Comparativement à une VM, un conteneur peut être démarré dans quelques secondes.
Espace disque	Les VMs utilisent beaucoup plus d'espace disque, car la totalité du noyau du système d'exploitation et les programmes qui y sont associés doivent être installés et exécutés	Les conteneurs utilisent peu d'espace disque, car ils partagent le même noyau du système d'exploitation que le système hôte.

1.3 Docker

Docker est une technologie libre de virtualisation basée sur les conteneurs. Ce projet, publié en logiciel libre (sous licence Apache 2.0), est l'initiative de Solomon Hykes et est supporté par une société américaine, nommée Docker. Docker est une plateforme ouverte pour les développeurs et les administrateurs systèmes (c.-à-d. le personnel d'infrastructure TI) et permet de construire, déployer, et exécuter des applications distribuées (Docker inc., 2015). En plus de bénéficier de toutes les caractéristiques de la technologie de virtualisation basée sur les conteneurs, Docker permet d'encapsuler une application, incluant toutes ses dépendances, dans un conteneur qui peut être ensuite exécuté sur n'importe quel autre système Linux. Ce qui rend la tâche beaucoup plus facile pour les développeurs et les administrateurs systèmes, car on n'a pas à se préoccuper de l'environnement physique sur lequel l'application va être exécutée.

Compte tenu la portabilité que procure Docker, il est devenu très populaire chez les professionnels d'infrastructures TI dans un laps de temps très court. Docker est très approprié pour être utilisé dans l'approche infonuagique Plateforme en tant que Service (PaaS). Selon

Polaert, c'est la technologie qui va révolutionner le monde de la virtualisation (2014). Google, IBM, RedHat, Microsoft, VMware et bien d'autres géants de l'informatique ont tous décidé de participer au projet (Polaert, 2014). Il faut aussi noter qu'il y a beaucoup d'efforts qui sont en en train de se faire de la part des grands acteurs informatiques pour rendre Docker disponible sur d'autres systèmes d'exploitation. Oracle a déjà annoncé le 30 juin 2015, son plan d'introduire Docker dans la prochaine version officielle du Système d'Exploitation Solaris. En somme, on est passé de machine physique à VM, puis actuellement, de VM à conteneur tel que Docker qui améliore la notion de portabilité d'une application.

Au cours de ces deux dernières années, l'intérêt pour la virtualisation à base de conteneur a particulièrement connu une croissance fulgurante (Linux Foundation, 2015). Tous les grands acteurs informatiques travaillent à mettre à la disposition du public leurs propres solutions à base de conteneur. Cette croissance peut s'expliquer surtout par l'importance de cette technologie dans l'approche informatique en nuage qui lui aussi, est un sujet d'actualité (Han et Jin, 2011). Devant cette prolifération de solutions basées sur les conteneurs, l'organisme Linux Foundation a lancé le 22 juin 2015 le «Open Container Initiative » (OCI), qui est une initiative regroupant 35 géants informatiques, visant à établir des normes autour de la création des conteneurs et de leur environnement d'exécution. Docker a offert son format de conteneur pour être servi comme base essentielle à cette initiative.

1.4 Gestion des ressources

À travers toutes ces évolutions, la gestion des ressources de l'hôte demeure toujours une perspective problématique à ne pas négliger. Qu'en est-il de la gestion des ressources avec ces nouvelles technologies? Dans le contexte de ce projet de recherche, on entend par gestion de ressources « l'allocation efficace, à chaque environnement virtualisé, des ressources adéquates pour bien accomplir sa tâche ». Une gestion efficace des ressources peut aider à accomplir trois buts importants pour l'administrateur de système (VMware, 2015):

1. Isolation de performance : empêcher les machines virtuelles de monopoliser les ressources et garantir un taux de service prévisible ;

2. Utilisation efficace : exploiter les ressources sous engagées et prendre des engagements excédentaires avec une dégradation gracieuse ;

3. Administration simple : contrôler l'importance relative des machines virtuelles, offrir un partitionnement dynamique souple et répondre aux accords de niveau de service absolu.

1.5 Méthodologie de recherche

Afin de réaliser des essais d'allocation de ressources à des conteneurs Docker, requis tout au long de ce projet de recherche, la version 4.3.24 de la technologie de virtualisation VirtualBox d'Oracle a été installée sur une plateforme Windows 7 (voir à l'annexe I, p.10). Un premier objectif est de créer une machine virtuelle sur laquelle sera mis en place un environnement Linux, de la distribution Ubuntu version 15.04. Par la suite, Docker 1.5 (voir à l'annexe II, p.11) est installé afin de permettre la création et la manipulation de conteneurs Docker utilisés au cours de l'expérimentation en laboratoire. C'est à l'aide de différentes commandes Docker que les diverses ressources (c.-à-d. processeur, mémoire, swap, disque, interface réseau et bien d'autres) seront manipulées afin de simuler différents scénarios d'allocation de ressources à des conteneurs Docker.

Le comportement des conteneurs et l'efficacité des procédures d'allocation de ressources seront ainsi évalués en appliquant des trafics élevés sur les conteneurs. Pour ce faire, différents outils Linux vont être utilisés tels que : « stress-ng » (voir à l'annexe IV, p.14) pour appliquer les trafics ; et « htop », « iotop » « systemd-cgtop » (voir à l'annexe IV, p.14) pour visualiser l'utilisation des ressources du système.

Conclusion

En conclusion on a vu que la virtualisation basée sur la technologie de type hyperviseur partitionne les composants du serveur physique afin de les présenter en composants virtuels à chaque VM, sur laquelle on peut installer un SE complet. Tandis que la virtualisation basée sur la technologie à base de conteneur utilise la même instance du SE, mais partitionne l'environnement d'exécution en créant des espaces d'exécution isolés. Docker est une plateforme libre faisant partie de la technologie de virtualisation à base de conteneur. Les techniques d'allocation de ressources fournies par le mode d'emploi en ligne de Docker seront utilisées dans cette recherche pour effectuer la gestion des ressources des conteneurs et

différents outils Linux vont être utilisés pour simuler et visualiser des trafics sur les conteneurs.

La prochaine section du rapport, tout d'abord, introduit les fonctionnalités « Espaces de noms » et « Groupes de contrôle » du noyau Linux, puis évoque brièvement comment elles contribuent dans l'architecture des conteneurs Docker et enfin présente les différentes expérimentations (c.-à-d. allocation de ressources) effectuées dans le cadre de cette recherche.

CHAPITRE 2

Gestion des ressources

Tel qu'il a été introduit dans le chapitre précédent, l'objectif principal de la recherche est d'explorer et évaluer la gestion des ressources des conteneurs Docker car il est fondamental que les ressources du serveur hôte soient distribuées d'une manière à permettre à ce que chaque conteneur puisse exécuter efficacement ses tâches. Dans ce chapitre, qui est le cœur de ce projet de recherche, il est important de jeter un bref regard sur les « Espaces de noms » et les « Groupes de contrôle » Linux, puis expliquer comment Docker les utilise. Finalement, les expérimentations effectuées seront présentées.

2.1 Espaces de noms Linux (Linux namespaces)

Le chapitre précédent introduisait la technologie de virtualisation à base de conteneur. Cette technologie rend possible l'exécution de plusieurs applications isolées les unes des autres sur un seul noyau Linux. Cette isolation est rendue possible grâce à la fonctionnalité « Espace de noms » du noyau Linux, et plus spécifiquement l'espace de nom par identifiant de processus (PID) qui permet d'isoler les processus du système d'exploitation. Cette fonctionnalité a été lancée, en janvier 2008, suite à la publication de la version 2.6.24 du noyau Linux (LinuxFr.org, 2008). Cette nouvelle version a été conçue dans le but d'apporter une solution au problème d'isolation. Il est maintenant possible d'avoir, sur un même serveur, des processus ayant les mêmes identifiants sans générer de conflits. À l'intérieur de chaque espace vont être exécutés des processus contrôlés par le noyau cependant inaccessible des autres espaces en train d'être exécutés sur le serveur.

Un système Linux, au moment de son démarrage, lance le programme « *init* » ayant comme identifiant processus (PID) 1, qui par la suite démarrera tous les autres processus requis pour continuer le démarrage et maintenir le système d'exploitation dans un état actif. « *init* » est le premier processus lancé et qui obtient automatiquement le droit d'hériter de l'identifiant 1. Tous les autres processus qui vont être démarrés par la suite seront les processus enfants du

« PID 1 », c'est-à-dire, « *init* » contrôlera tous les autres processus du serveur. Il pourra les suspendre ou les stopper au besoin. Il existe cette même hiérarchie de processus à l'intérieur d'un conteneur, car les conteneurs ne sont que des instances isolées du système d'exploitation du serveur hôte. Dans une approche traditionnelle, les processus ne peuvent pas être dupliqués, car il ne peut pas exister plusieurs processus ayant le même identifiant sur un même hôte. L'approche de « Espace de noms» rend possible cette cohabitation grâce à sa propriété de créer des environnements isolés. Cela permet, entre autres, au noyau de gérer plusieurs arborescences de processus. Par exemple, il ne pourra pas confondre le répertoire racine(/) du conteneur A à celui du conteneur B, ou du moins à celui du serveur hôte. Néanmoins, cette approche ne se résume pas seulement au niveau de processus, mais aussi au niveau de ressources. Il existe plusieurs types d'espaces de noms. Le tableau ci-dessous présente une très brève description de chaque type d'espace de nom.

Tableau 2.1 Différents types d'espaces de noms (Kerrisk, 2015)

Espace nom	**Constant**	**Isolation**
IPC	CLONE_NEWIPC	Communication interprocessus tels que System V IPC, messages POSIX, etc. …
Network	CLONE_NEWNET	Ressources réseau
Mount	CLONE_NEWNS	Point de montage
PID	CLONE_NEWPID	Identifiants processus
User	CLONE_NEWUSER	Espaces utilisateurs tels que les identifiants de groupes et d'utilisateurs
UTS	CLONE_NEWUTS	Identifiants systèmes « nodename » et « hostname »

2.2 Groupes de contrôle (cgroups)

Les groupes de contrôle Linux : « cgroups » offrent une fonctionnalité fournie par le noyau du système d'exploitation Linux permettant la gestion et la surveillance des ressources du système d'exploitation, y compris le temps processeur, la mémoire de système, les entrées-sorties de disques et la bande passante réseau. « Ce mécanisme permet l'agrégation et le partitionnement des tâches du système et tous leurs futurs enfants, en groupes hiérarchiques

avec un comportement spécialisé » (kernel.org, 2015). Tout comme la fonctionnalité « espace de noms », elle a été lancée avec la version 2.6.24 du noyau Linux.

Sous-systèmes

Paul Menage décrit, dans la documentation de « cgroups » disponible sur le site officiel de l'organisme kernel.org, un sous-système à titre de module utilisant les propriétés fournies par les groupes de contrôle qui consiste à grouper des tâches avec l'objectif de les manipuler de façons particulières. Un sous-système est aussi un contrôleur de ressources. Les sous-systèmes disponibles dans la distribution Linux «Ubuntu 15.04» sont les suivants (Canonical Ltd, 2015) :

- cpusets: Utilisé pour assigner des processeurs spécifiques et des nœuds de mémoire à un groupe de contrôle ;
- blkio : Sert à fixer des limites sur l'utilisation des blocks entrée-sorties ;
- cpuacct : Produit des rapports sur l'utilisation des ressources processeurs ;
- devices: Contrôle l'habilité des taches dans un groupe de contrôle pour créer ou utiliser des périphériques ;
- freezer: Permet de suspendre ou de relancer des taches dans un groupe ;
- hugetlb : Utilisé pour contrôler l'utilisation de hugetlb dans un groupe ;
- memory: Permet de limiter l'utilisation de la mémoire vive et du swap ;
- net_cls: Permets d'identifier les paquets réseau par leur identifiant de classe. Ces informations sont ensuite utilisées par le contrôleur de trafic (tc) Linux ;
- net_prio : Pour établir des priorités sur les trafics réseau sur une base de groupe ;
- cpu : Utilise le planificateur pour établir la priorité d'exécution des tâches ;
- perf_event : Fais une surveillance par processeur pour seulement les threads dans un groupe quelconque.

Les groupes de contrôle peuvent être utilisés de différentes manières :

- En utilisant des commandes shell de Linux telles que : mkdir, mount etc… ;
- En utilisant des utilitaires comme cgcreate, cgexec, cgclassify, etc… ;

- Par le biais des logiciels de virtualisation à base de conteneur comme LXC et Docker.

2.2.1 Comment les groupes de contrôle peuvent être utiles

Prenons un scénario hypothétique ou il y a plusieurs applications en exécution sur un serveur. Il y a une des applications qui est particulièrement gourmandes et il est nécessaire de limiter sa consommation à 15GB de mémoire, 60% d'entrée-sorties disque et 50% de temps CPU. Grâce à la fonctionnalité « cgroups » il est possible de créer un groupe de contrôle Linux avec des limitations de consommation de ressources précises et assigner l'application à ce groupe. L'application une fois ajoutée à ce groupe ne pourra plus dépasser la quantité de ressources attribuées au groupe lorsqu'il y a contention des ressources du serveur. Cependant, l'application pourra quand même aller au-delà des limites fixées quand toutes les ressources ne sont pas en utilisation (c.-à-d. qu'il y a de la capacité non utilisée). Tout cela peut se faire dynamiquement sans avoir à redémarrer le serveur hôte. Les groupes de contrôle utilisent le même principe de hiérarchie que les processus. Le comportement d'un attribut d'un parent peut alors influencer un groupe de contrôle enfant, car ils héritent certains attributs de leur parent. Cependant, contrairement aux processus, plusieurs hiérarchies de groupe peuvent exister sur un même système.

2.3 L'Architecture de Docker

Docker possède une architecture client-serveur qui se compose : d'un client Docker et d'un démon qui joue le rôle de serveur (Docker Inc., 2015). Le client, qui est l'interface utilisateur principale de Docker, utilise les commandes de Docker pour lancer ses requêtes au serveur. Le serveur (démon Docker) à son tour se sert des paramètres fournis par le client et communique au noyau du système d'exploitation Linux par le biais des pilotes d'exécution LXC ou « libcontainer » pour construire, exécuter et distribuer les conteneurs Docker. Le démon Docker utilise les espaces de noms et les groupes de contrôle Linux pour l'isolation et la répartition de ressources de ses conteneurs. Lancé initialement avec le pilote d'exécution LXC qui nécessite l'interface d'espace utilisateur LXC pour manipuler les «espaces de

noms» et les «groupes de contrôle», Docker, à partir de la version 0.9 utilise « libcontainer » comme pilote d'exécution par défaut (Docker, 2014). Il est possible de passer d'un pilote à l'autre mais cependant, il est conseillé de ne pas exécuter sur une hôte qui utilise le pilote « libcontainer » un conteneur créé dans un environnement utilisant le pilote lxc et vice versa. « Changer le pilote d'exécution peut affecter considérablement la façon dont Docker interagit avec le noyau et cela peut introduire des dépendances d'exécution sur les conteneurs» (Mathias et Kane, 2015). Les expérimentations effectuées au cours de ce travail de recherche sont faites dans un environnement utilisant le pilote « libcontainer ». Pour l'instant, Docker est compatible seulement avec le système d'exploitation Linux. Cependant, cela est sur le point d'évoluer rapidement. La société Microsoft, a récemment fait part de son intention de rendre disponible Docker sur le très populaire système d'exploitation Windows. Par la suite, la compagnie Oracle Corporation a annoncé, le 30 juillet 2015, qu'elle compte très prochainement intégrer Docker dans le système d'exploitation Unix Solaris.

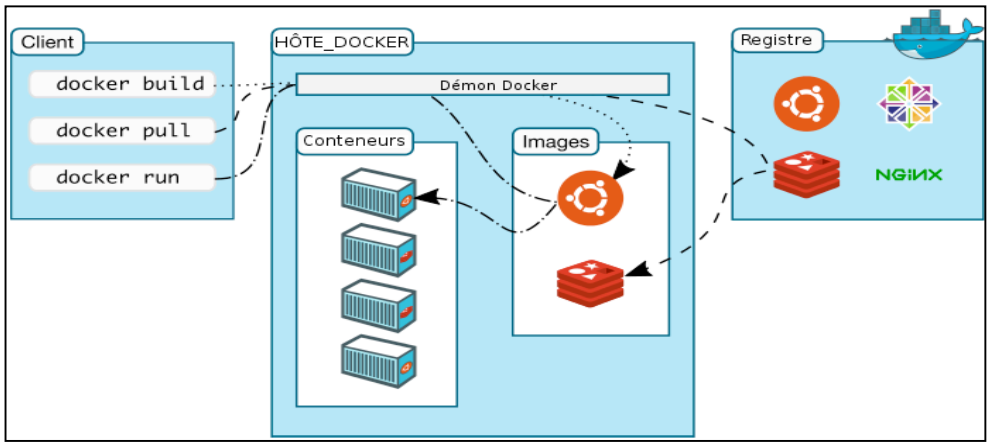

Figure 2.1 Représentation de l'architecture client-serveur de Docker (Docker Inc., 2014)

2.4 Docker et l'utilisation des ressources

Tel qu'il a été discuté au chapitre 1, Docker est une technologie de virtualisation à base de conteneur. Cette approche s'est distinguée principalement par sa manière d'utiliser les

ressources. Cette technologie diffère de la technologie à base d'hyperviseur qui nécessite le partitionnement des ressources physiques d'un serveur hôte afin de pouvoir installer un système d'exploitation isolé avec toutes ses dépendances. Docker, de son côté, utilise la même instance du SE pour exécuter ses conteneurs, ce qui a pour résultat une utilisation des ressources beaucoup plus efficace.

Au cours de ce projet de recherche, Oracle VM VirtualBox a été utilisé afin de configurer un environnement d'essais contenant une machine virtuelle de 30G de disque et 2Gb de mémoire sur lequel la distribution Linux « Ubuntu Desktop Edition » a été installée. Il est publié qu'avec Oracle VM VirtualBox (c.-à-d. comme pour plusieurs autres technologies de virtualisation à base d'hyperviseur) qu'il est possible de faire de l'allocation granulaire (c.-à-d. de réserver des ressources dont la capacité physique sera réellement allouée au fur et à mesure que l'on l'utilise). Il est observé que juste pour l'installation du système d'exploitation, 512MB de mémoire vive et 8GB d'espace disque comme est initialement requis (Canonical Ltd, 2015), afin de permettre le bon fonctionnement du SE, sans tenir compte des ressources nécessaires pour les autres applications. Dans un environnement d'entreprise utilisant cette technologie, ce scénario se répèterait pour autant de VM qu'on aurait créée. Donc, cette grande quantité de ressources serait utilisée seulement pour les SE. Cependant, avec un conteneur Docker qui utilise pratiquement les mêmes librairies du SE hôte, cela aurait complètement changé la donne. Un conteneur Docker aurait utilisé juste quelques mégaoctets d'espace disque pour sauvegarder les détails de sa configuration. Par exemple, dans l'environnement d'essais configuré pour effectuer les expérimentations nécessaires tout le long de ce projet de recherche, l'image Docker Debian utilisée pour créer les conteneurs exécutant le système d'exploitation Debian Linux occupe seulement 125.2MB d'espace disque et celle de MySQL 282.9MB.

2.4.1 Docker et l'utilisation de certaines fonctionnalités du noyau Linux

Docker utilise les fonctionnalités «Espace de noms» et «Groupe de contrôle» pour isoler ses conteneurs. Du coup, cela évite à un utilisateur la complexité d'utiliser les lignes de

commandes Linux pour créer un environnement isolé utilisant les espaces de noms. À ce stade du développement de Docker, il existe encore certaines limitations, Docker ne permet pas officiellement l'utilisation de tous les sous-systèmes et les espaces de noms (Docker Inc., 2015), par exemple l'espace de nom utilisateur, mais il peut être manipulé directement utilisant les commandes du SE Linux.

2.5 Limitation des ressources avec Docker

À travers cette section vont être présentées les différentes expérimentations qui ont été faites dans l'environnement laboratoire implémenté afin d'effectuer les essais requis pour cette recherche, tel qu'il a été présenté au chapitre 1 du rapport à la section 1.6. Pour chacun des essais d'allocation de ressources, des outils Linux ont été utilisés pour tester et valider les valeurs allouées. Les résultats de ces observations seront, par la suite, publiés au chapitre 3 du rapport.

2.5.1 Ressources mémoires

Cette section présente les différentes démarches qui ont été utilisées pour effectuer des essais d'allocation de mémoire sur des conteneurs.

2.5.1.1 Mémoire vive (RAM) et mémoire virtuelle (SWAP)

Les options -m ou --memory passées à la commande « docker run » au moment de la création d'un conteneur permettent de limiter la mémoire du conteneur (Docker Inc., 2015). Tel qu'il a été mentionné précédemment dans ce rapport, à la section 2.2, cette limitation de ressources est possible grâce à la fonctionnalité « cgroups » du noyau Linux. Il faut donc activer le contrôleur de mémoire vive et de mémoire virtuelle des « cgroups » (voir détails à l'annexe II) afin de pouvoir utiliser cette option. Il faut aussi noter que l'activation du contrôleur de mémoire du groupe de contrôle peut avoir un effet négatif sur la performance du système hôte, car le noyau devra effectuer un contrôle détaillé sur l'utilisation de la mémoire (Docker Inc., 2015). C'est ce qui explique que dans la majorité des distributions Linux cette fonctionnalité est par défaut désactivée (c.-à-d. swapaccount=0).

Pour effecteur cette expérimentation, une image Docker de la distribution Linux Debian a été téléchargée. Cette image a été ensuite utilisée pour créer un conteneur auquel 512 mégaoctets de mémoire vive ont été alloués. Cette configuration permettra d'avoir un SE dans lequel on pourra faire différents essais pour expérimenter le comportement du conteneur par rapport à la limitation de mémoire. Par défaut, la même valeur allouée à la mémoire vive est aussi assignée à la mémoire virtuelle. Cependant, s'il devient nécessaire de changer cette valeur par défaut, il faudra le faire en utilisant l'option « memory-swap » qui prend comme paramètre la valeur combinée de la mémoire vive et la mémoire virtuelle. On peut aussi désactiver la limitation mise sur la mémoire virtuelle en configurant « memory-swap » à -1 (en passant l'option « --memory-swap -1 » à la commande « docker run »), ce qui signifie que le conteneur peut utiliser le swap du serveur hôte autant qu'il y a de disponible. Toutes ces options ont été expérimentées sur plusieurs conteneurs en utilisant comme base la même image publique Docker Debian téléchargée sur le site de partage d'images de Docker https://github.com . Après la création des conteneurs, la commande « docker inspect » est utilisée pour voir si l'allocation de ressources a été faite correctement. On peut aussi se servir de la commande « docker stats » qui affiche la quantité de mémoire allouée et utilisée par le conteneur durant son exécution.

2.5.1.2 L'option Out of Memory (--oom-kill-disable)

Le noyau Linux alloue de la mémoire à la demande des applications qui exécutent sur le système (Kernel.org, 2015). Couramment, certaines applications font un sur approvisionnement mémoire, mais n'utilisent pas la totalité de cette mémoire ainsi allouée et ce permet au noyau d'allouer plus de mémoire qu'il y en a de disponible sur le serveur hôte. Lorsque les applications commencent à utiliser réellement la mémoire, le noyau, pour préserver son exécution, tue automatiquement certaines tâches. Pour gérer ce comportement du noyau, plusieurs options ont été fournies parmi lesquelles *--oom-kill-disable* dont la syntaxe peut être légèrement différente selon la distribution de Linux utilisée. Cette fonctionnalité de base, du noyau Linux, est aussi rendue disponible dans Docker à partir de la version 1.7 (Docker inc., 2015). Pour configurer notre conteneur Debian, avec cette option, Docker a été mis à jour à la version 1.7 (voir à l'annexe III), car elle n'étais pas disponible

dans la version 1.5 installée au début du projet de recherche. Il est précisé, sur le site de Docker, d'utiliser cette option sur un conteneur dont la mémoire est limitée (c.-à-d. option *-m* ou *--memory*) afin d'éviter que le conteneur utilise toutes les ressources RAM et SWAP du serveur hôte.

2.5.2 Ressource processeur

La même image Docker Debian a été utilisée pour faire des essais d'allocation de temps processeur. Au cours de cette activité, plusieurs conteneurs ont été créés dans le but de réaliser les expérimentations suivantes : 1) allouer un processeur dédié à un conteneur ; 2) partager un ou plusieurs processeurs entre des conteneurs ; 3) définir la durée que les processus d'un conteneur peuvent s'exécuter dans un intervalle précis.

2.5.2.1 Allocation de CPU spécifique (option --cpuset-cpus ou --cpuset dans les versions antérieures à 1.7)

Selon l'application qui s'exécute dans un conteneur, il peut survenir des situations où il est nécessaire d'établir la capacité de traitement à un nombre fixe de processeurs. Ceci peut être réalisé en utilisant l'option « --cpuset-cpus » lors de la création du conteneur. L'option « --cpuset-cpus », anciennement connue sous l'étiquette de « --cpuset », mais qui a été renommée à partir de la version 1.7 de Docker, permet d'assigner un ou plusieurs processeurs à un conteneur s'exécutant sur un hôte multicœur (Docker Inc., 2015). Elle prend comme paramètre d'entrée les identifiants processeur disponibles sur le serveur hôte, où chaque cœur, ou « thread », du serveur physique est présenté au conteneur comme un processeur virtuel. On peut donc établir un intervalle (par exemple --cpuset-cpus="0-4"), permettant l'identification de processeurs spécifiques (par exemple --cpuset-cpus="0, 2, 6") peut être aussi précisés.

L'environnement de laboratoire, ayant été configuré initialement avec un seul processeur, a été modifié afin d'y ajouter un autre processeur dans le but d'effectuer ces essais. Ainsi le nombre de processeurs, après la reconfiguration, est de deux, avec les identifiants 0 et 1.

Dans un premier temps, un conteneur a été créé auquel l'identifiant processeur 1 a été assigné. Une fois créé, la commande Linux « nproc » a été utilisée pour valider la configuration processeur de ce conteneur. Afin d'observer le comportement du conteneur crée quand il y a un usage intensif des ressources processeurs, «stress-ng» a été installé dans ce conteneur afin de produire des tâches consommant du temps processeur. Dans un second temps, la même expérience a été répétée, toutefois les deux identifiants processeurs 0 et 1 ont été alloués au conteneur.

2.5.2.2 Configuration de limite d'utilisation processeur relative (--cpu-shares ou –c)

Le but visé de cet essai était de contrôler la pondération de la puissance de traitement de plusieurs conteneurs. L'option --cpu-shares ou –c, passée à la commande «docker run», lors de la création d'un conteneur, permet d'atteindre ce but. Un « share », qui peut se traduire en par le mot « partie » ou « portion », définit une portion du temps processeur du système alloué à une tâche (Oracle Corporation, 2012). Cette fonctionnalité est implémentée dans les systèmes Solaris en utilisant le concept de « Fair Share Scheduling » (FSS). Dans un système Linux, la valeur par défaut est de 1024, et il en est ainsi pour les conteneurs Docker (Docker Inc., 2015). Considérée toute seule, la valeur de la « part de cpu » n'a aucune importance. Il faut aussi tenir compte du nombre de conteneurs qui s'exécutent sur le serveur hôte et aussi de la valeur de la « portion de cpu » qui leur est assignée, car une tâche dont la valeur de la « portion de cpu » est fixée à zéro va quand même s'exécuter, même lorsque le processeur est totalement libre. Les scénarios suivants présentent deux situations dans lesquelles l'option --cpu-shares est utilisée afin de contrôler les ressources processeurs de plusieurs conteneurs.

Scénario 1

Prenons l'exemple d'un serveur hébergeant quatre conteneurs configurés avec l'option --cpu-shares et ayant des valeurs différentes. Supposons que le conteneur C1 est configuré à 1024 « portions de cpu »; le deuxième, C2 et le troisième C3, ont une valeur de 512 pour chacun; et le quatrième, C4 est fixé à une valeur de 256 « portions de cpu ». Lorsqu'il y a une contention de ressources et que les processus des quatre conteneurs tentent d'utiliser 100%

des ressources processeur, C1 consommerait 44% des ressources processeur total ; C2 et C3 prendraient chacun 22% ; et C4 utiliserait 11%.

Scénario 2

Supposons à présent qu'il y a que deux conteneurs qui s'exécutent sur le serveur hôte. Un conteneur C1, dont la limite de « portions de cpu » est fixée à 75 et un autre conteneur C2 auquel une valeur de 50 « portions de cpu » a été allouée. Dans ce cas, quand il y a contention de ressources processeurs, le conteneur C1 utilisera 60% des ressources tandis que le conteneur C2 utilisera une proportion de 40%.

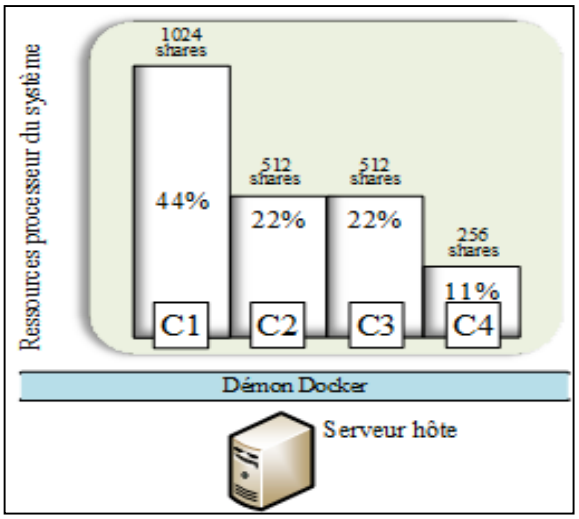

Figure 2.2 Partage des ressources CPU entre 4 conteneurs hébergés sur le même serveur hôte et configurés avec l'option cpu-shares

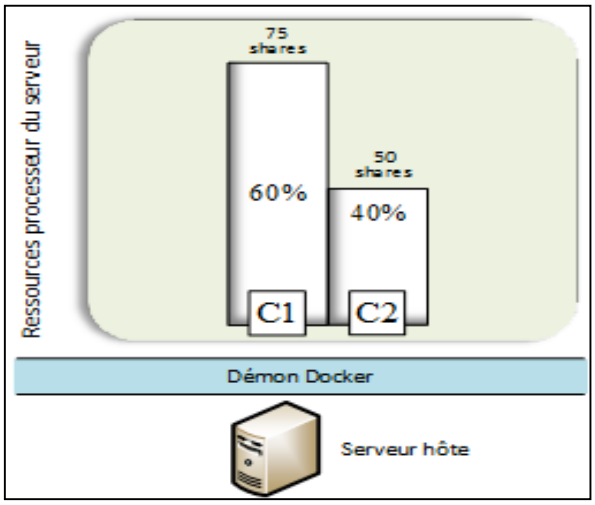

Figure 2.3 Partage des ressources CPU
entre 2 conteneurs hébergés sur le
même serveur hôte et configurés
avec l'option cpu-shares

À la lumière de tout ce qui a été exposé dans cette section, pour réaliser cette expérimentation il est nécessaire d'avoir au moins deux conteneurs. Donc, deux conteneurs ont été créés avec l'option --cpu-shares auxquels des valeurs de 512 et de 1024 « parts de cpu » ont été assignées. Pour s'assurer d'une contention de ressources processeur imminente, l'option «--cpuset-cpus » a été aussi utilisée pour les assigner au même processeur. Après la création des conteneurs, tout d'abord « docker inspect » est utilisé pour vérifier les valeurs « part de cpu » allouées à chaque conteneur. Ensuite, l'outil Linux « stress-ng » a été installé dans chaque conteneur dans le but d'appliquer du trafic consommant de la puissance processeur. Finalement, les outils « systemd-cgtop » et « htop » ont permis de valider la proportion de processeur utilisée par chaque conteneur.

2.5.2.3 Configuration de limite d'utilisation processeur absolue (--cpu-quota et --cpu-period)

Les options «--cpu-quota » et «--cpu-period», rendues disponibles à partir de la version 1.7 de Docker, se présentent aussi comme une alternative visant à limiter l'utilisation du temps processeur d'un conteneur. Contrairement à l'option «--cpushares», «--cpu-quota» et «--cpu-

period» permettent de fixer une limite absolue à l'utilisation des ressources processeur, ce qui signifie que la consommation de ressources des processus d'un conteneur configuré avec ces options ne pourra jamais dépasser la limite fixée ; indépendamment s'il y a des ressources inutilisées ou pas. Ces fonctions exploitent les fonctionnalités de l'ordonnanceur de ressources processeur par défaut « Completely Fair Scheduler » (CFS) (qui se traduit en français par Ordonnanceur Complètement Équitable), du Système d'Exploitation Linux. Cette fonctionnalité permet de spécifier la quantité maximum de bande passante processeur disponible à un groupe ou une hiérarchie (Kernel.org, 2015). «--cpu-quota» et «--cpu-period» sont contrôlés par le sous-système cpu du groupe de contrôle. La valeur cpu par défaut est établie à -1, ce qui indique qu'il n'y a aucune restriction de bande passante sur ce groupe de contrôle. La valeur de 100ms indique la durée d'une période, alors que «--cpu-period» spécifie l'intervalle de temps, en microsecondes, auquel les ressources processeur alloué à un conteneur doivent être réattribuées. L'option «--cpu-quota» indique la quantité totale de temps, en microsecondes, pour laquelle les tâches d'un conteneur peuvent s'exécuter pendant une période (Docker, 2015). Cette quantité de temps, une fois écoulée, toutes les tâches du conteneur seront mises en attente en attendant la prochaine période.

Pour effectuer cette expérimentation, deux essais ont été réalisés. Le premier consistait à créer un conteneur configuré de manière à ce que ses processus ne puissent utiliser plus de 50% des ressources processeurs du système hôte. Il a donc été configuré avec des valeurs de «--cpu-period» et «--cpu-quota» respectives de 100000 (c.-à-d. 100 secondes) et 50000 (c.-à-d. 50 secondes). Ceci signifie que, pour chaque période de 100 secondes de temps processeurs, les processus du conteneur peuvent utiliser jusqu'à 50 secondes. Cette expérience a été faite sur une hôte munie d'un seul processeur.

Un second essai a aussi été réalisé et visait à créer un conteneur dont les tâches utilisent jusqu'à 15% des ressources processeur d'un serveur hôte configuré avec deux processeurs, lorsqu'il y a contention de ressources processeur. Pour y arriver, le conteneur a été créé de manière à ce que ses processus ne puissent utiliser que 15 secondes (c.-à-d. --cpu-quota=15000) au cours d'une période de 100 secondes (c.-à-d. –cpu-period=100000) de

temps processeur. Encore une fois, la commande Docker «docker inspect» a été utilisée pour vérifier les nouvelles valeurs attribuées aux options «--cpu-quota» et «--cpu-period» lors de la création des conteneurs, puis «stress-ng» a été installé dans les conteneurs pour simuler des trafics consommant beaucoup de ressources processeur.

2.5.3 Gestion de la bande passante des entrées-sorties et de l'espace de stockage de disque dur

Dans cette section, les essais pour limiter l'accès à la bande passante des entrées-sorties de disque dur et aussi contrôler l'espace de stockage dans un conteneur sont réalisés et décrits. Encore une fois, les mêmes images Docker Debian sont réutilisées pour créer les conteneurs. La première étape de l'expérience consiste à limiter l'accès à la bande passante du disque dur à un conteneur et la seconde étape vise à permettre à un conteneur d'utiliser une quantité d'espace supérieure à celle allouée par défaut et aussi de pouvoir sauvegarder les données après qu'il ait été stoppé.

2.5.3.1 Gestion de l'utilisation de la bande passante de disque

Les ressources de la machine hôte sont limitées, y compris le disque dur. Il a un nombre limité de requêtes en lecture ou en écriture qu'il peut recevoir et traiter par seconde (IOPs). Au-delà de cette quantité, il y aura une file d'attente et cela impactera nécessairement la performance du serveur hôte et, conséquemment, de tous les autres conteneurs qu'il héberge. Donc, pour bien gérer la performance d'une application, il est incontournable de connaitre les exigences de chaque application, en termes d'opérations d'entrées-sorties disque. La performance des applications est donc liée à la quantité d'accès disque qu'elles peuvent effectuer dans une période de temps donnée, par exemple, les opérations de lectures et écritures sur une base de données MySQL ou Oracle.

Prenons l'hypothèse d'un serveur hôte hébergeant plusieurs dizaines de conteneurs avec chacun ses exigences en matière d'opérations entrées-sorties disque et qu'il faut limiter leur accès à la bande passante de disque. Docker, avec son pilote par défaut, ne propose pas

beaucoup d'options, quoiqu'on puisse recourir à d'autres moyens pour assurer cette gestion d'une manière plus complète (c.-à-d. manipulation directe des groupes de contrôle avec systemctl ou utilisation des options lxc). Parue à la version 1.7 de Docker, en avril 2015, l'option «--blkio-weight» permet d'assigner une limite relative d'opérations d'entrées-sorties disque à un conteneur. Cette option est gérée par le sous-système «blkio» de la fonctionnalité cgroups et peut être configurée avec une valeur comprise entre 10 et 1000 (Docker Inc, 2015). La valeur par défaut alloué à un conteneur est de 0 ce qui signifie que les processus du conteneur n'ont aucune limite quant à l'accès des opérations entrées-sorties disque. Tel qu'il a été mentionné précédemment, l'option «--blkio-weight» permet d'établir une limite relative qui est l'équivalent de l'option «--cpu-shares» pour les opérations entrées-sorties disque.

Afin d'expérimenter le comportement de cette option, deux conteneurs ont été créés et configurés avec des valeurs «--blkio-weight» différentes. Un premier auquel une valeur de 400 a été allouée et un second dont la valeur a été fixée à 100. «docker inspect» a été utilisé pour confirmer les nouvelles valeurs assignées; puis avec l'outil «Linux dd», des essais chronométrés, simulant la copie d'une même quantité de blocs de disque sur chaque conteneur, ont été effectués afin d'établir l'impact des valeurs assignées au «--blkio-weight» sur l'utilisation de la bande passante entrées-sorties d'un conteneur.

2.5.3.2 Gestion de la capacité de stockage d'un conteneur

Un autre aspect très important de la gestion des ressources d'un conteneur est le contrôle de sa capacité de stockage. Au cours des paragraphes précédents, les conteneurs et les applications ont été présentés, mais très peu ou presque pas des données ont été traitées dans ces conteneurs, ce qui ne reflète pas une utilisation réelle. Alors qu'en est-il du traitement de données? Comment, à travers Docker, gérer et manipuler l'espace de stockage des données des conteneurs ? «La base de l'utilisation des fichiers systèmes dans Docker est-ce qu'on appelle l'abstraction du «backend storage» (Project Atomic, 2015) qui, traduit en français, est le «stockage centralisé» ou «stockage dorsal». Les images Docker sont stockées dans des

couches de fichiers disponibles seulement en lecture. Lors du démarrage d'un conteneur, à partir d'une image existante, une autre couche est ajoutée au-dessus de l'image, et ce en lecture et en écriture. Si le conteneur modifie un fichier existant, ce fichier est sauvegardé au plus haut niveau de la couche disponible en lecture et écriture (Mouat, 2015). Toutes ces couches sont prises en charge par un fichier système capable de créer l'interface entre Docker et le système d'exploitation hôte. C'est le «stockage dorsal». Le «stockage dorsal» permet de sauvegarder un ensemble de couches qui peuvent être adressées chacune par un nom unique (Project Atomic, 2015). Les types de «stockages dorsaux» (c.-à-d. fichiers systèmes) les plus populaires et supportés par Docker sont les suivants : vfs, devicemapper, btrfs et aufs qui est considéré comme le «stockage dorsal» original de Docker (Project Atomic, 2015). Le type de stockage «aufs» est celui utilisé par défaut dans la distribution Ubuntu Linux configurée pour ce travail de recherche. Cependant, si nécessaire, il serait simple de passer d'un type à un autre en reconfigurant le démon Docker (c.-à-d. le fichier /etc/default/docker) avec l'option « --storage-driver», qui prend comme paramètre le type de «stockage dorsal».

S'il est nécessaire de sauvegarder des données persistantes ou de manipuler de grands volumes de données, ce qui nécessite une quantité considérable d'opérations entrées-sorties disque, Docker recommande l'utilisation des «volumes de données» et les «conteneurs de volumes de données » (Docker Inc., 2015).

Une première expérimentation à cet effet, effectuée et décrite au cours de cette section est de créer un conteneur auquel un «volume de données» est assigné. Un «volume de données» est un répertoire ou un fichier système hébergé sur le serveur hôte, mais accessible à un ou plusieurs conteneurs; et, du coup, évitant la structure constituée des couches du «stockage dorsal» de Docker. Cette approche offre l'avantage d'être indépendante du cycle de vie du conteneur et manipulable à partir du serveur hôte au même titre que n'importe qu'elle autre répertoire ou fichier système. L'option –v ou «VOLUME» passée aux commandes «docker run» ou «docker create» permet d'ajouter un «volume de données» à un conteneur. Elle prend comme paramètres d'entrée les points de montage du fichier système sur le serveur hôte et le conteneur. Cette option peut être utilisée à multiples reprises pour spécifier plusieurs fichiers systèmes à monter dans un conteneur. Par défaut, cette option monte les

volumes en mode lecture et écriture, cependant ce comportement peut être modifié en ajoutant «:ro» (c.-à-d. en mode lecture seulement) à la fin du point de montage.

La seconde consiste à créer un «conteneur de volume de données» et l'utiliser comme point de stockage pour un autre conteneur applicatif. Un conteneur de volume de données n'est autre qu'un conteneur stoppé utilisant des «volumes de données» dans le but de les rendre disponibles à d'autres conteneurs. L'un des objectifs de cette approche est de centraliser le stockage des données. Pour utiliser un «conteneur de volume de données», il suffit de passer à la commande «docker run» le nom du «conteneur de volume de données» en utilisant l'option «--volumes-from».

2.5.3.3 Gestion de la bande passante réseau

Il serait tout aussi important de pouvoir limiter la consommation de bande passante réseau d'un conteneur, car tout comme pour les ressources processeur, mémoire et autres, un conteneur qui utilise de façon abusive la bande passante de l'interface réseau peut causer une dégradation ou une interruption de service. Ainsi, il est nécessaire, autant que possible, d'isoler les conteneurs, car toutes les applications ont chacune leurs exigences en matière de ressources. Certaines applications utilisent beaucoup de bande passante disque tandis que d'autres nécessitent, de préférence, une plus grande proportion de temps processeur ou de bande passante réseau. Par exemple un conteneur hébergeant un serveur web qui reçoit une quantité importante de requêtes d'utilisateurs ne demande pas la même quantité de bande passante réseau qu'un conteneur jouant le rôle d'un serveur de fichier interne à une organisation.

Cependant, pour l'instant, Docker n'offre pas de solutions pouvant aider à contrôler les ressources réseau. Supposons qu'il y a une interface réseau de 10Ghz installée sur un serveur hôte sur lequel s'exécutent des conteneurs, et que l'on veut limiter à 2Ghz un conteneur susceptible d'utiliser une grande quantité de bande passante réseau. Il n'y a aucune option disponible dans Docker pour accomplir cette tâche. Plusieurs groupes d'intérêts spécialisés ont été consultés pour voir l'évolution des recherches effectuées pour compenser ces options non disponibles dans la version courante de Docker. Au moment de rédiger ce rapport,

aucune solution documentée n'est disponible. Étant en pleine évolution, ce besoin va être sans doute abordé dans les prochaines versions de Docker. En attente d'une solution, les outils «pipework», «tc» et les sous-systèmes de groupe de contrôle «net_cls» et «net_prio» sont en train d'être expérimentés par plusieurs groupes spécialisés afin de voir comment ils peuvent être combinés ensemble afin de contrôler la bande passante réseau dans les conteneurs Docker.

Conclusion

Ce chapitre a présenté, de manière succincte, en premier lieu, les technologies «espaces de noms» et les «groupes de contrôle» du système d'exploitation GNU Linux qui permettent en arrière-plan l'isolation et la gestion des ressources des conteneurs Docker. Les «espaces de noms» rendent possible l'exécution de plusieurs applications isolées les unes des autres sur un seul noyau Linux tandis que les «groupes de contrôles» permettent la répartition et la surveillance de certaines ressources du système d'exploitation Linux, tels que le processeur, la mémoire vive, la mémoire swap et le disque dur. En deuxième lieu, l'architecture sommaire de Docker et le lien existant entre Docker et certaines fonctionnalités Linux ont été introduits. En dernier lieu, les différentes expérimentations de gestion de ressources des conteneurs Docker effectuées dans le cadre de la recherche ont été présentées.
Le chapitre suivant présente les résultats des expérimentations et essais qui ont été introduits dans ce chapitre.

CHAPITRE 3

Présentation des résultats

Ce chapitre présente les résultats des différentes expérimentations visant les allocations de ressources des conteneurs Docker qui ont été effectuées au cours de ce travail de recherche.

3.1 Limitation de la mémoire

3.1.1 Mémoire vive (RAM) et mémoire virtuelle (SWAP)

L'expérimentation effectuée pour valider la limitation de mémoire démontre que les 512Mo passés à l'option -m de la commande «docker run» lors de la création du conteneur ont été automatiquement configurés au niveau du groupe de contrôle créé pour permettre l'isolation des ressources, comme la valeur de la mémoire vive. Une vérification manuelle du fichier «memory.limit_in_bytes», situé dans le répertoire spécifique du conteneur à l'intérieur de /sys/fs/cgroup/memory/system.slice, retourne cette valeur en octet (c.-à-d. 536870912 octets). La commande «docker inspect» a permis de consulter des informations concernant le conteneur. La figure 3.1 présente la sortie d'écran de la commande «docker inspect».

La figure 3.2 présente, de son côté, le graphe d'utilisation de la mémoire RAM lorsque l'outil «stress-ng» est utilisé pour charger des pages dans la mémoire. Premièrement, il est possible de déduire que même lorsqu'il y a une grande consommation de la RAM, à l'intérieur du conteneur, les tâches qui s'exécutent ne pourront pas dépasser la limite des 512 Mo de mémoire vive impartie (voir la Figure 3.2), et cela ne pourra pas affecter les autres conteneurs et le serveur hôte non plus. Néanmoins, dans le cas où il y a des demandes de réservation de mémoire impossibles à satisfaire à cause d'un manque de ressources (c.-à-d. une sur allocation), les tâches du conteneur qui consomment beaucoup de ressources sont automatiquement détruites afin de libérer de la mémoire (voir la Figure 3.3), et ceci ne prend pas en compte si le conteneur a été créé avec l'option «--oom-kill-disable» ou pas. Ce qui ne devrait être le cas si cette option est désactivée (c.-à-d. --oom-kill-disable=false), car selon la documentation de Docker, le comportement par défaut (c.-à-d. --oom-kill-disable=true) serait

de tuer les tâches utilisant beaucoup de mémoire RAM quand il y a une sur allocation. L'observation qui a été faite lors de l'expérimentation démontre plutôt que c'est le noyau du serveur hôte qui tue les tâches lorsqu'il y a une sur allocation étant donné que le noyau gère le groupe de contrôle dans lequel s'exécute le conteneur.

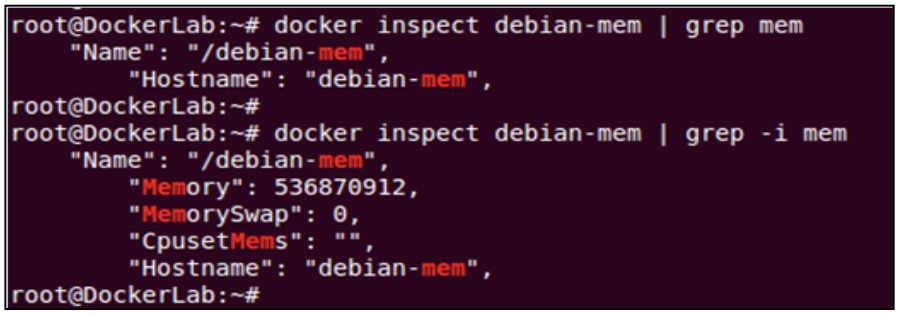

Figure 3.1 Sortie d'écran de la commande «docker-inspect» exécutée pour vérifier la quantité de mémoire allouée au conteneur

Deuxièmement, il est possible de constater que lorsque les tâches du conteneur atteignent la limite de la mémoire RAM fixée, elles commencent à utiliser la mémoire swap, ce qui est une situation normale. «stress-ng» permet de charger des pages en mémoire qui pourraient réserver jusqu'à 1024 Mo, c'est-à-dire, la valeur combinée de la mémoire RAM et la mémoire vive du conteneur. Une fois cette limite atteinte, on observe que les tâches «stress-ng» sont automatiquement tuées et un message OOM est enregistré dans les journaux du système hôte et du conteneur. Il est aussi important de limiter l'utilisation du swap, car le fait de donner à un conteneur le libre accès d'utiliser tout le swap disponible sur le serveur hôte provoque un ralentissement de tout le système dû à un manque de ressources mémoire pour sauvegarder les pages mémoires susceptibles à être réutilisées dans un temps proche.

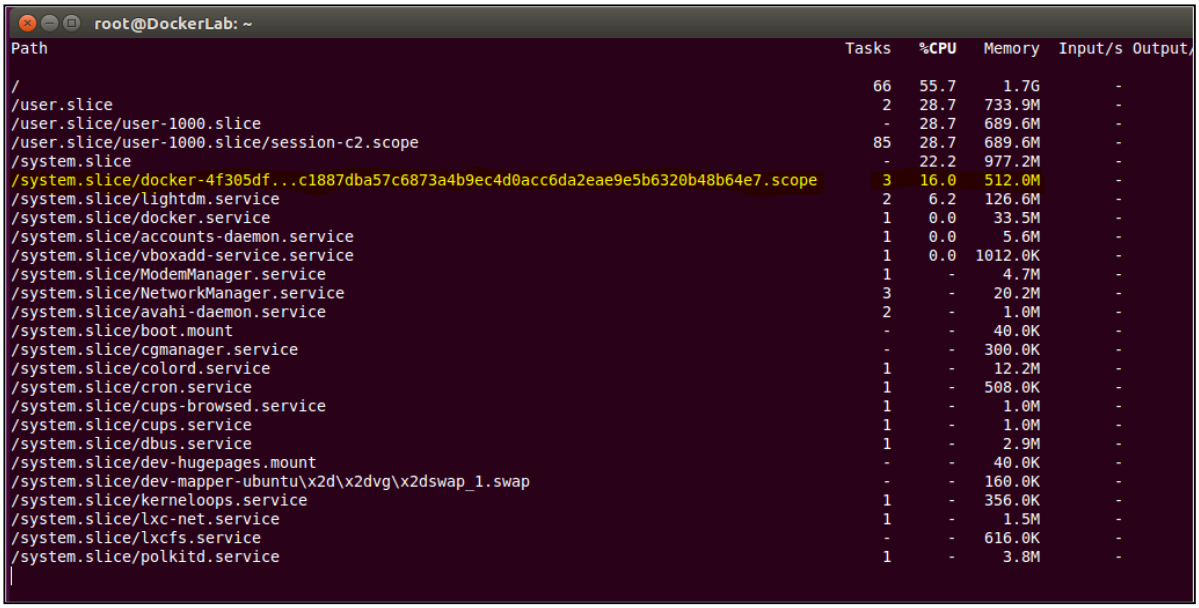

Figure 3.2 Copie d'écran de la commande «systemd-cgtop» affichant la quantité de mémoire consommée par le conteneur Docker

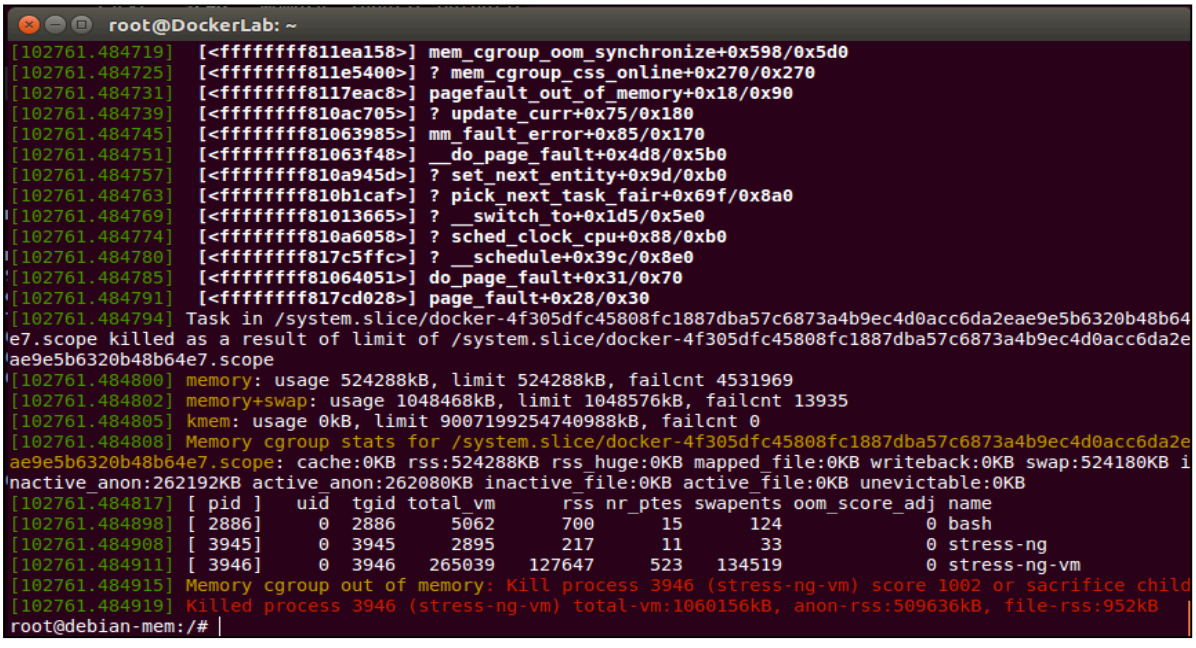

Figure 3.3 Extrait du journal log après qu'un évènement «OOM killer» ait été produit

3.2 Limitation des ressources processeur

3.2.1 Allocation de CPU spécifique

Les figures 3.4 et 3.6 présentent le graphe de la consommation des ressources processeur d'un conteneur auquel des processeurs spécifiques ont été alloués. L'exécution des tâches dans les conteneurs est simulée avec l'outil «stress-ng» et comme on peut le constater dans ces deux figures, les deux premiers processus en exécution démontrent les deux modules d'essais de charge de traitement processeur lancé à l'intérieur du conteneur. Il est à constater que la consommation de la puissance de traitement processeur du conteneur se limite évidemment au processeur dédié quel que soit la quantité de la charge de traitement assignée. Le processeur 1 (c.-à-d. identifié par le noyau par processeur 0) dédié au conteneur est utilisé à sa capacité maximale tandis que l'autre processeur du serveur hôte reste à environ 97% inutilisé (voir la Figure 3.4). Les processus en exécution dans le conteneur partagent à priorité égale le temps processeur disponible. Tel que représenté à la figure 3.4, chaque module d'essais consomme 50% du temps processeur étant donné que deux modules seulement ont été lancés. Cela est une indication, sans que l'on puisse l'affirmer, que c'est une manière efficace de s'assurer qu'un conteneur n'utilise pas plus de temps processeur qu'il en faut surtout avec les applications qui ne relâchent pas les ressources après les avoir réservé.

Conséquemment, il a été possible de comprendre qu'un même processeur peut être dédié à différents conteneurs, c'est-à-dire qu'il n'y a aucune restriction de réutiliser un même identifiant processeur, autant de fois que possible. Cependant quand il y contention de ressources processeur, les tâches vont partager à parts égales la puissance de traitement du processeur. Quand le même identifiant processeur est assigné à un deuxième conteneur sur lequel la même charge de traitement a été lancée, il a été possible d'observer que chaque tâche consomme 25% du temps processeur (contrairement au scénario précédent où elles utilisaient 50% chacune) étant donné que quatre tâches sont en exécution.

Cela permet aussi qu'on puisse assigner tous les processeurs du serveur hôte à un ou plusieurs conteneurs comme cela a été le cas décrit par la figure 3.6. Les deux seuls

processeurs de la VM ont été alloués à un conteneur, les essais de charge de traitement, une fois lancés, à l'intérieur du conteneur causent une lenteur au niveau du serveur hôte qui pourrait affecter tous les conteneurs hébergés sur le serveur. Une situation à laquelle qu'il serait bon d'éviter.

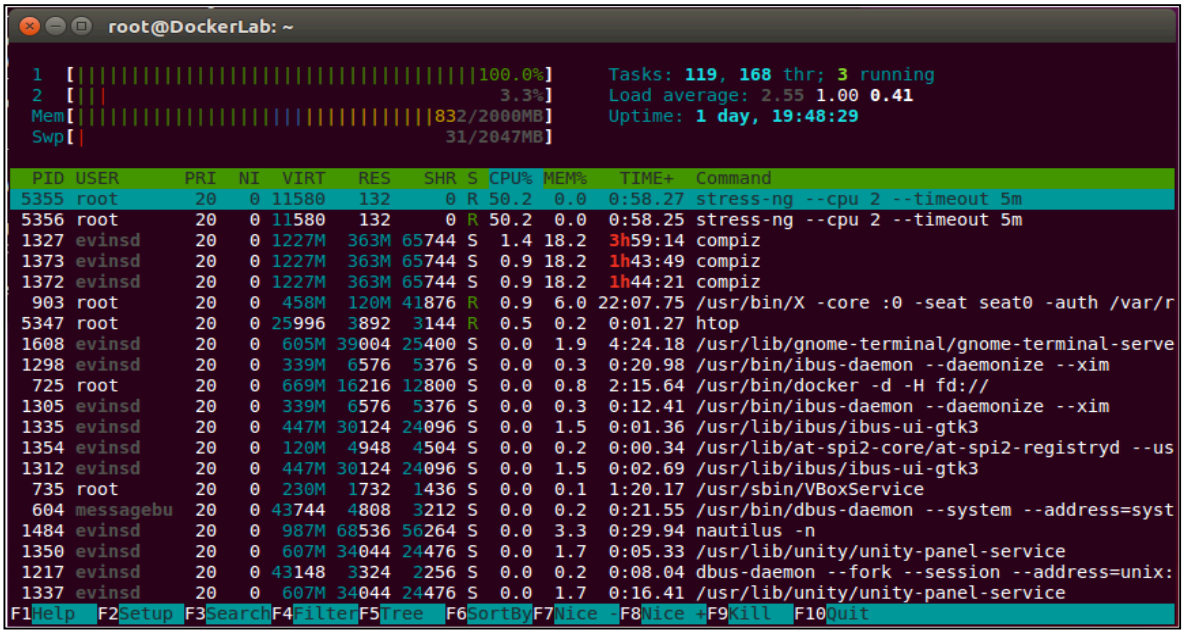

Figure 3.4 Capture d'écran affichant le graphe d'utilisation du processeur ID 0

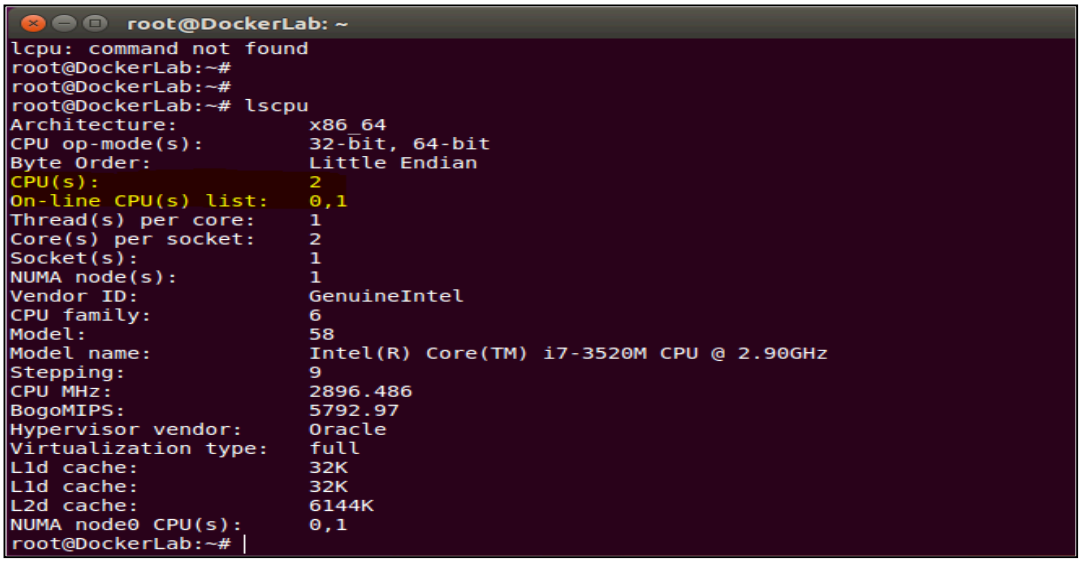

Figure 3.5 Capture d'écran affichant les identifiants du processeur disponibles sur le serveur hôte

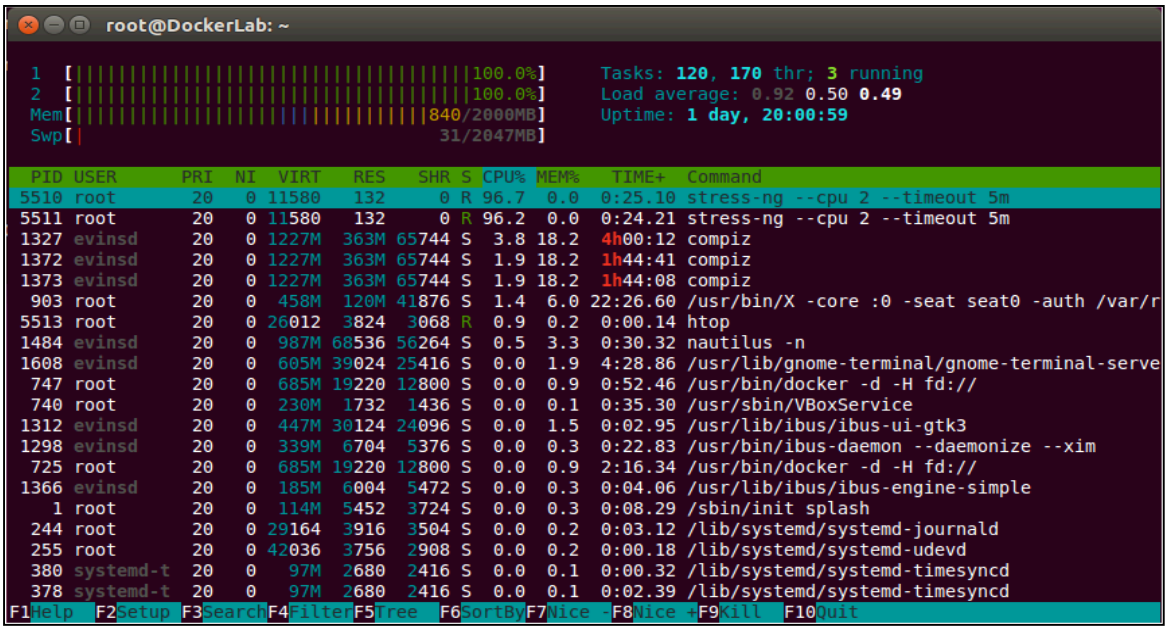

Figure 3.6 Capture d'écran affichant le graphe d'utilisation des deux processeurs du serveur hôte tous deux assigné au conteneur

3.2.2 Configuration de limite d'utilisation relative de traitement processeur

Tel qu'il a été présenté à la section 2.5.2.2 du chapitre 2, l'expérimentation a été réalisée à l'aide de deux conteneurs. La figure 3.7 illustre le pourcentage de ressources processeur consommé pour chaque tâche lancée dans les conteneurs lors des essais avec l'outil «stress-ng», tandis que la figure 3.8 présente le pourcentage du temps processeur utilisé par chaque groupe de contrôle dans lequel s'exécutent les conteneurs. Il est possible de constater, dans un premier temps, que pour les quatre modules d'essais lancés (voir la Figure 3.7), deux d'entre eux consomment chacun environ 33% des ressources du processeur 1 auquel il est lié. Ces deux tâches s'exécutent à l'intérieur du conteneur dont, 1024 parts cpu, ont été alloués. Les deux autres tâches utilisent chacune 16.5% du temps processeur, dont la moitié de la proportion consommée par le conteneur précédent, étant donné que 512 parts cpu lui ont été assignés, dont la moitié de la quantité allouée au conteneur précédent.

Il a été aussi observé, dans un second temps, que les tâches du conteneur, fixées à 512 parts de cpus, commencent à consommer chacune 50% des ressources processeur une fois les essais sur l'autre conteneur terminé. Cela explique le fait que l'option –c ou –cpushare fixe seulement une limite d'utilisation relative et que le comportement varie en fonction du nombre de conteneurs et de tâches en exécution utilisant les processeurs en question.

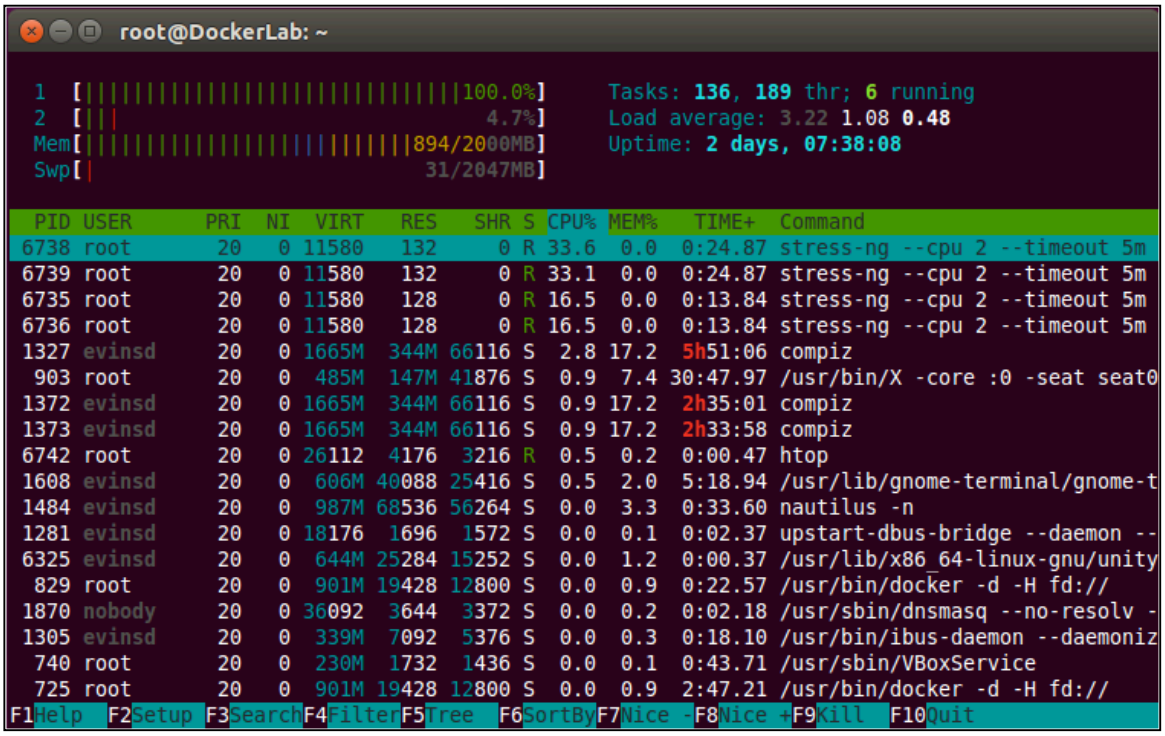

Figure 3.7 Sortie d'écran de la commande «htop» affichant la proportion de temps processeur utilisé par chaque tâche stress-ng lancée

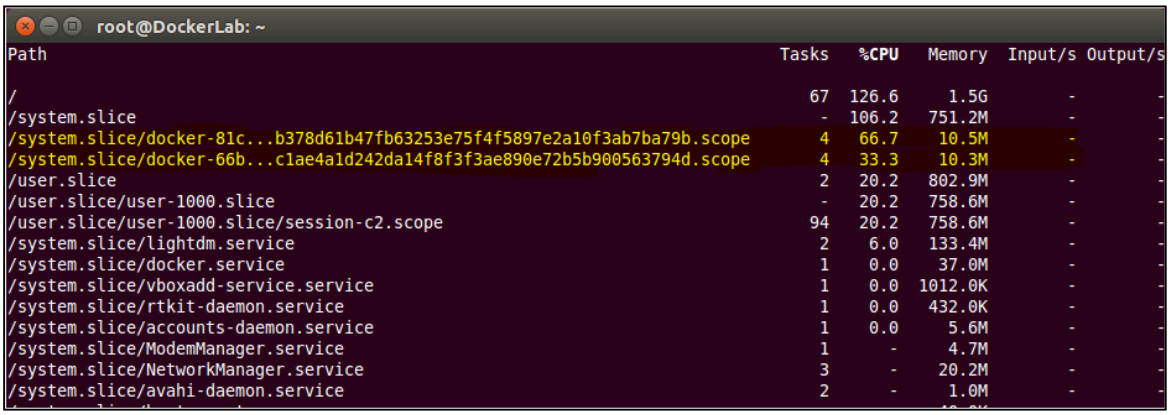

Figure 3.8 Sortie d'écran de la commande «systemd-cgtop» affichant la proportion

de temps processeur utilisé par chaque groupe de contrôle dans lequel s'exécute les deux conteneurs.

En utilisant l'outil «dd» avec la commande Linux «time» pour effectuer, sur chaque conteneur en même temps, un essai qui copie des blocs de 512 octets 100 mille fois dans un fichier, il a été remarqué que le conteneur auquel 1024 parts de processeur ont été assignées complète cette opération en 20.9878 secondes. Ce conteneur réussit à copier une moyenne de 2.4 mégaoctets par seconde (voir la Figure 3.10). Tandis que l'autre conteneur, configuré avec seulement 512 parts de processeurs, complète cette même opération en 9 secondes de plus et copie une moyenne de 1.7 mégaoctet par seconde (voir la Figure 3.9). Ce scénario illustre, sans ambigüité, qu'en cas de contention de ressources processeur le conteneur paramétré avec un plus grand nombre de parts de cpu utilise plus de puissance de traitement par rapport à un autre configuré avec un nombre de parts de cpu inférieures.

```
root@debian-cpushare0:/#
root@debian-cpushare0:/# time dd if=/dev/zero of=/root/testfile bs=512 count=100000 oflag=direct
100000+0 records in
100000+0 records out
51200000 bytes (51 MB) copied, 29.9592 s, 1.7 MB/s

real    0m29.961s
user    0m0.024s
sys     0m12.512s
root@debian-cpushare0:/#
```

Figure 3.9 Capture d'écran du résultat de l'essai effectué dans le conteneur 1 auquel 512 shares ont été alloués

```
root@debian-cpushare1:/#
root@debian-cpushare1:/# time dd if=/dev/zero of=/root/testfile bs=512 count=100000 oflag=direct
100000+0 records in
100000+0 records out
51200000 bytes (51 MB) copied, 20.9878 s, 2.4 MB/s

real    0m21.158s
user    0m0.024s
sys     0m9.616s
root@debian-cpushare1:/#
```

Figure 3.10 Capture d'écran du résultat de l'essai effectué dans le conteneur 2 auquel 1024 shares ont été alloués

3.2.3 Configuration de limite d'utilisation processeur absolu

Contrairement à l'approche précédente, l'approche d'essai vise à établir une limitation maximale de l'utilisation des ressources processeur d'un conteneur. La consommation du temps processeur des tâches du conteneur ne devraient pas dépasser cette limite fixée. Les figures 3.11, 3.12 et 3.15 illustrent le pourcentage du temps processeur consommé par des conteneurs. Tel que présenté au chapitre 2, la validation et l'efficacité de cette approche ont été possibles en créant deux conteneurs. Le premier possède des tâches qui sont destinées à utiliser un maximum de 50% des ressources processeur du serveur hôte et le second dont la consommation de ses tâches pourrait atteindre jusqu'à 15% des ressources processeur global du système.

Cette expérimentation démontre que lorsqu'il y a seulement une tâche d'essai en exécution dans le conteneur, elle consomme jusqu'à 50% du temps processeur qui lui a été assignée et se limite directement à un processeur. Ce comportement constitue un scénario prévisible étant donné qu'il n'y a pas de contention de ressources processeur et la tâche ne peut pas être fragmentée. Elle utilise donc 50% d'un seul processeur de la VM qui héberge le conteneur et il a été observé que l'autre processeur est à environ 96% inutilisé. Cette observation est facilement validée à l'aide de «htop » qui affiche le graphe d'utilisation processeur de la tâche en exécution et de «systemd-cgtop» qui affiche le pourcentage des ressources consommées par le groupe de contrôle dans lequel s'exécute le conteneur (voir la Figure 3.12), incluant le processeur. Cependant, il a été remarqué que : quand 500 tâches ont été lancées, et que tous ces processus s'activent pour du temps processeur, la charge est répartie à parts égales entre les deux processeurs (voir la Figure 3.14) alors que le pourcentage d'utilisation du temps processeur du conteneur conserve toujours la même valeur de 50%. Les mêmes observations ont été faites avec le conteneur auquel a été assigné 15% des ressources processeur du serveur hôte (Figure 3.15). Toutes ces expériences confirment, sans contredit, que les options «--cpu-quota» et «--cpu-period» sont utiles afin de fixer la consommation maximale de ressources processeur d'un conteneur Docker.

En outre, il a été remarqué que, quand les tâches en exécution sont nombreuses, elles ne s'exécutent pas en même temps. L'exécution est coordonnée par le CFS Linux comme sur n'importe quel autre système. Les figures 3.11 et 3.15 démontrent respectivement 422 et 491 tâches en exécution dans les conteneurs alors que 500 ont été lancées. Le quota de temps processeur pour une période de temps est aussi géré par le CFS en fonction du nombre de tâches en exécution. Cela prouve ainsi qu'avec ce procédé il n'est pas permis de limiter un conteneur à un processeur fixe ou à un groupe de processeurs. La limite maximale de ressource processeur assigné au conteneur est donc fonction de la capacité de traitement globale du serveur hôte.

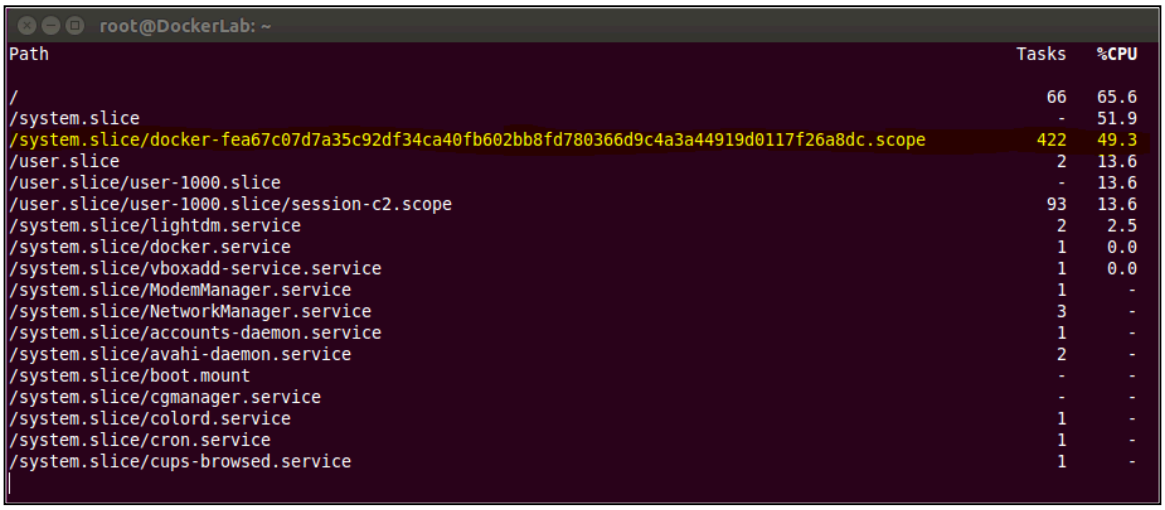

Figure 3.11 Capture d'écran affichant le pourcentage de la puissance processeur utilisé par 500 taches d'essais lancées dans le conteneur avec stress-ng

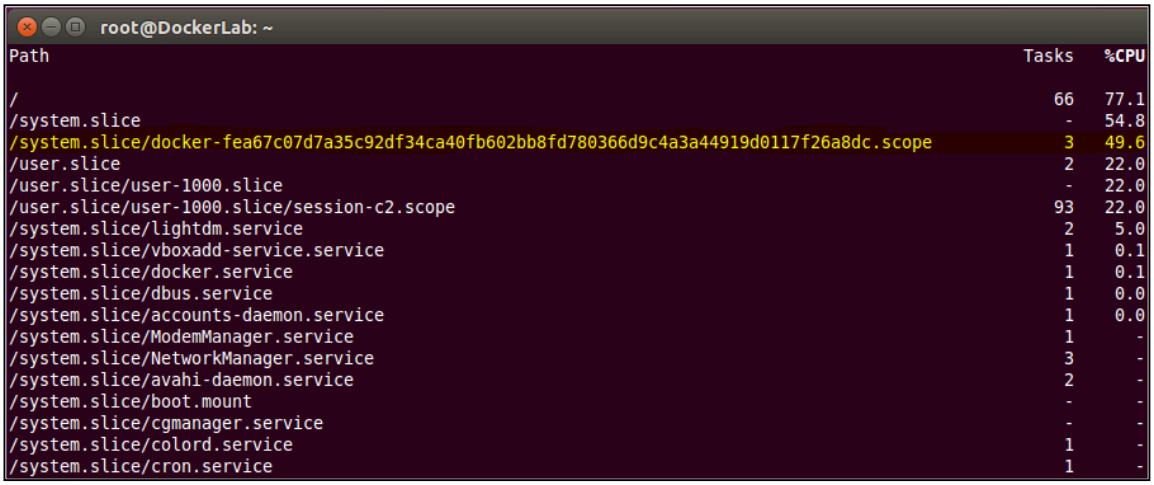

Figure 3.12 Capture d'écran affichant le pourcentage de la puissance processeur utilisé par 1 tache lancée dans le conteneur avec l'outil stress-ng

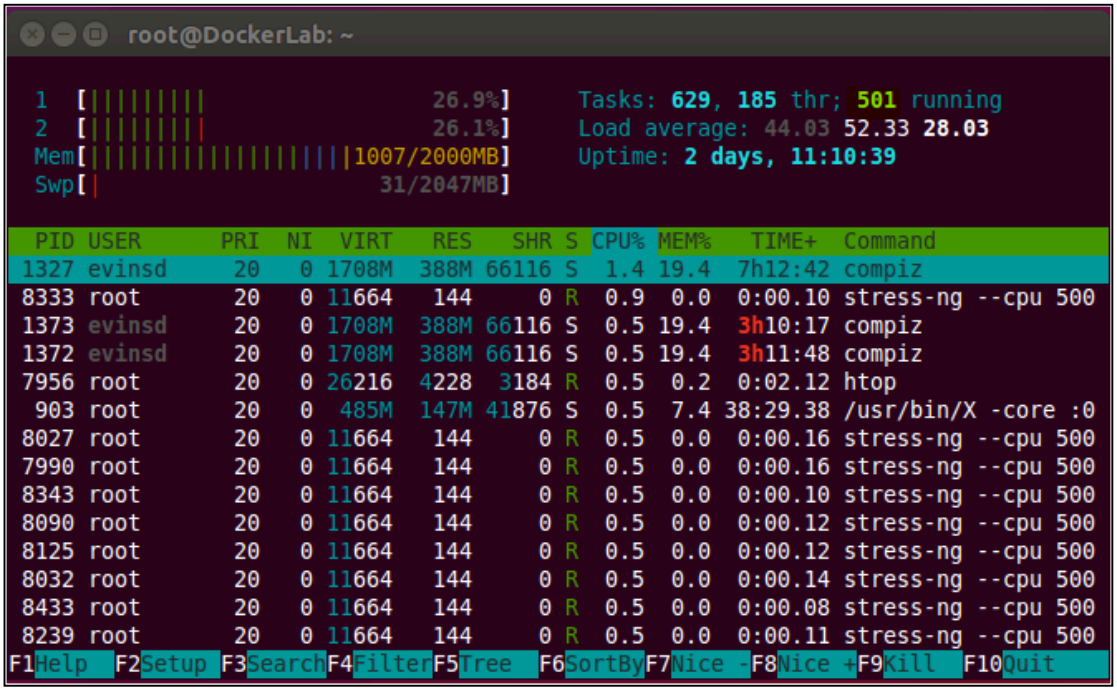

Figure 3.14 Capture d'écran illustrant environ 74% de chaque processeur inutilisé alors que 500 tâches d'essai sont en exécution

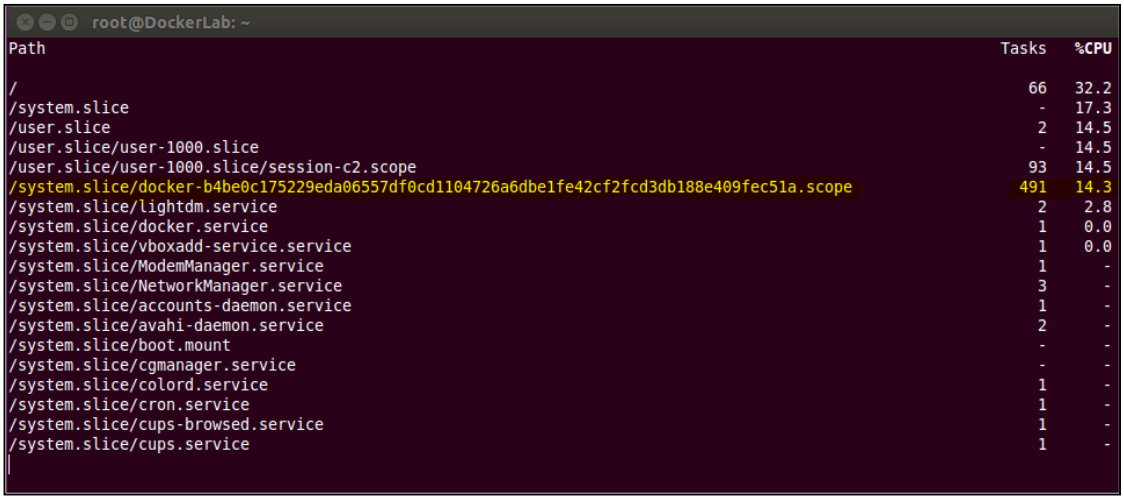

Figure 3.15 Capture d'écran illustrant environ 15% d'utilisation de ressources processeur par le conteneur alors que 500 tâches d'essai sont lancées.

3.3 Limitation de l'utilisation des entrées sorties et de l'espace de stockage disque

3.3.1 Gestion des de l'utilisation de la bande passante disque

Les figures 3.16 et 3.17 présentent les essais effectués sur les deux conteneurs auxquels des valeurs respectives de 1000 et 250 ont été assignées à l'option «--blkio-weight». Ces essais consistent à copier 100000 fois des blocs de 1024 octets dans un fichier, et ils ont été répétés plusieurs fois dans le but de bien valider les résultats obtenus. Premièrement, il a été constaté que les copies utilisent approximativement la même quantité d'IOPS lorsqu'elles sont lancées en même temps, soit 3.6 mégaoctets par seconde (Mb/s) pour le conteneur dont la valeur a été configurée 1000 et 3.5 (Mb/s) pour l'autre conteneur configuré à 250. Alors que l'option «--blkio-weight» du premier conteneur est configurée avec une valeur équivalente à quatre fois plus du second. Deuxièmement, la valeur «real» retournée par la commande Linux «time» qui affiche la durée de temps réel passé jusqu'à la complétion de la commande reste encore une fois presque la même, 28.06 secondes et 28.951 secondes respectivement pour le premier et le second conteneur. Cependant, d'après la documentation de Docker ce temps devrait être proportionnel à la valeur assignée à l'option «--blkio-weight» de chaque conteneur.

Ces résultats aident à déduire que l'utilisation de cette option pour gérer la limitation de la bande passante du disque dure ne donne pas les résultats escomptés.

Figure 3.16 Capture d'écran présentant illustrant la copie dans un fichier des blocs de disque de la taille de 1024 octets 100 mille fois dans un conteneur 1 auquel l'option «--blkio-weight» a été configurée à 1000

Figure 3.17 Capture d'écran illustrant la copie dans un fichier des blocs de disque de la taille de 1024 octets 100 mille fois dans un conteneur 2 auquel l'option «--blkio-weight» a été configurée à 250

3.3.2 Gestion de la capacité de stockage

Les figures 3.18 et 3.19 illustrent les volumes de données hébergés sur le serveur hôte et montés dans le conteneur. Il a été observé dans un premier temps, que seulement un volume est apparu dans la sortie de la commande «df –h» exécutée à l'intérieur du conteneur et qui affiche les fichiers systèmes visibles du conteneur. Pour des raisons non identifiées, quand

plusieurs volumes sont montés dans le conteneur, un seul volume d'entre eux est visible avec la commande «df», et pour voir tous les volumes et les fichiers systèmes montés dans le conteneur il faut les lister avec la commande Linux «ls» (figure 3.18). Dans un second temps, il a été aussi observé que les répertoires ou fichiers systèmes présentés dans le conteneur sont accessibles à partir du serveur hôte avec les mêmes permissions et toutes les modifications effectuées sur l'hôte sont instantanément accessible dans le conteneur. Et en dernier lieu, tel qu'il a été prévu toutes les données persistent après la destruction du conteneur. Ce qui permet comme l'a mentionné Docker de garder ces données intactes, indépendantes de la vie des conteneurs qui perdent toutes les données stockées dans la structure du stockage dorsal lors de son exécution.

En outre, le conteneur dans lequel les volumes sont montés a été présenté à un autre conteneur comme un conteneur de volumes de données. Les constats effectués montrent que tous les volumes montés préalablement dans le conteneur sont donc visibles dans le nouveau conteneur (Figure 3.10) et peuvent être affichés à l'aide de la commande «df». Il est aussi important de souligner que le conteneur de volume de données ne nécessite pas d'être en exécution pour qu'il puisse être présenté à un autre conteneur. Toutes les expériences montrent de manière irréfutable que les données des conteneurs gérées à travers les volumes de données et conteneurs de volumes de données sont préservées lorsque le conteneur est détruit et ces volumes peuvent être présentés à plusieurs conteneurs en même temps.

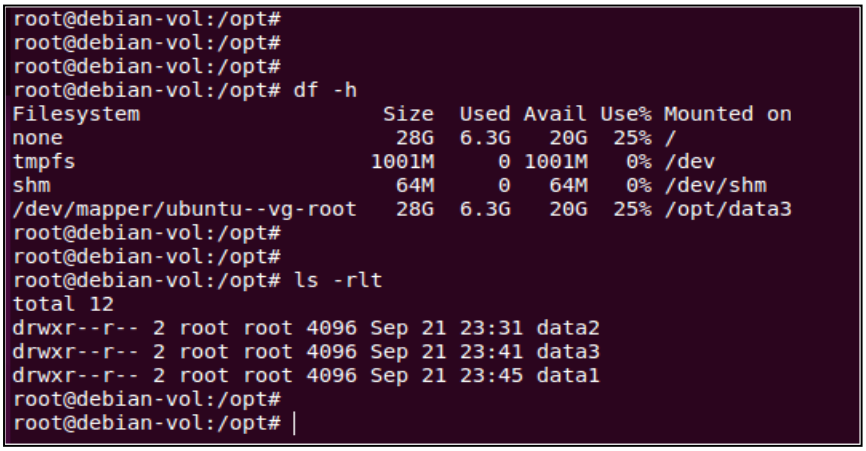

Figure 3.18 Capture d'écran affichant trois volumes présentés à un conteneur

```
root@debian-vol:/opt# dd if=/dev/zero of=/opt/data2/test2 bs=1G count=8 oflag=direct
8+0 records in
8+0 records out
8589934592 bytes (8.6 GB) copied, 122.332 s, 70.2 MB/s
root@debian-vol:/opt#
root@debian-vol:/opt#
root@debian-vol:/opt# ls -rlt
total 12
drwxr--r-- 2 root root 4096 Sep 21 23:41 data3
drwxr--r-- 2 root root 4096 Sep 21 23:57 data1
drwxr--r-- 2 root root 4096 Sep 22 00:01 data2
root@debian-vol:/opt#
root@debian-vol:/opt#
root@debian-vol:/opt# du -sh *
7.1G    data1
8.1G    data2
4.0K    data3
root@debian-vol:/opt#
```

Figure 3.19 Capture d'écran affichant plus de 10Gb de données dans un conteneur gérés à partir de volumes de données

```
root@mysqlapp1:~# df -h
Filesystem                  Size  Used Avail Use% Mounted on
none                         28G  6.3G   20G  25% /
tmpfs                      1001M     0 1001M   0% /dev
shm                          64M     0   64M   0% /dev/shm
/dev/mapper/ubuntu--vg-root   28G  6.3G   20G  25% /opt/data3
/dev/mapper/ubuntu--vg-root   28G  6.3G   20G  25% /opt/data1
/dev/mapper/ubuntu--vg-root   28G  6.3G   20G  25% /opt/data2
/dev/mapper/ubuntu--vg-root   28G  6.3G   20G  25% /var/lib/mysql
/dev/mapper/ubuntu--vg-root   28G  6.3G   20G  25% /etc/resolv.conf
/dev/mapper/ubuntu--vg-root   28G  6.3G   20G  25% /etc/hostname
/dev/mapper/ubuntu--vg-root   28G  6.3G   20G  25% /etc/hosts
tmpfs                      1001M     0 1001M   0% /proc/kcore
tmpfs                      1001M     0 1001M   0% /proc/timer_stats
root@mysqlapp1:~#
root@mysqlapp1:~#
```

Figure 3.20 Capture d'écran illustrant le conteneur debian-vol précédant présenté comme un conteneur de volumes de données à un conteneur de base de données MySQL

Conclusion

Ce chapitre a présenté les résultats détaillés et les observations des expérimentations établies au chapitre 2 de ce rapport. Premièrement, les résultats des essais pour limiter la mémoire vive et la mémoire swap d'un conteneur démontrent que les options proposées par Docker offrent, sans contredit, les résultats escomptés, sauf pour l'option « --oom-kill-disable » qui épouse le même comportement qu'elle soit activée ou pas. Deuxièmement, la possibilité de définir des limites d'utilisation processeur relatives et absolues dans les conteneurs avec les options –c, --cpu-share, --cpu-quota et --cpu-period prouvent qu'elles offrent de bons moyens pour gérer les ressources processeur. Troisièmement, les résultats expérimentaux démontrent avec succès qu'il est bien possible de sauvegarder les données créées et modifiées par des conteneurs avec l'utilisation des volumes de données et les conteneurs de volumes de données. Les volumes de données sont des répertoires ou fichiers systèmes hébergés sur le serveur hôte, monté dans un conteneur lors de sa création.

Cependant, pour l'instant, Docker n'a pas d'option ou de solution pour gérer la bande passante réseau.

CONCLUSION

Ce travail de recherche a été réalisé dans le but d'expérimenter et faire une évaluation des options de commande proposées par Docker pour la gestion des ressources de ses conteneurs. La virtualisation, à base de conteneur, est une technologie émergente et peu d'information est disponible pour bien comprendre la gestion des ressources.

Les expérimentations de gestion de ressources effectuées au cours de cette recherche ont porté sur les aspects suivants :

Premièrement la mémoire (c.-à-d. RAM et SWAP). Les résultats obtenus de ces essais prouvent que la majorité des options fournies par Docker pour limiter les ressources mémoires se révèlent très efficaces. Il est possible de mettre une limite relative à la quantité de mémoire ou de swap qu'un conteneur peut utiliser quand il y a contention de ressources du serveur hôte, cependant il pourra utiliser autant de mémoire disponible lorsque les ressources sont inutilisées. Les ressources mémoires d'un conteneur peuvent être aussi fixées à une limite à laquelle il ne pourra pas dépasser une fois cette limite atteinte. Néanmoins, il a été constaté que l'option OOM killer disponible dans la version 1.7 de Docker devrait être revue par l'équipe, car elle ne fonctionne pas comme l'aurait documenté par Docker.

Deuxièmement, les ressources processeurs. Tel qu'il a été vu pour les ressources mémoire, les résultats montrent aussi que la limitation des ressources processeur dans les conteneurs Docker donne bien le résultat escompté en utilisant les options de Docker. Avec l'option –c, un conteneur peut être lié à un ou plusieurs processeurs spécifiques. Aussi, comme on a vu précédemment pour la mémoire, une limite d'utilisation relative de ressources processeur peut être fixée à un conteneur pour spécifier la quantité maximale de temps processeur à utiliser lorsqu'il y a contention de ressources. Il est aussi à noter qu'en plus de ces deux procédés, «cpu-quota» et «cpu-period » peuvent être utilisé pour fixer littéralement le quota de ressource processeur à utiliser pour une période de temps donné.

Et enfin la bande passante des opérations entrées-sorties du disque dur et la capacité de stockage. Les résultats des essais réalisés montrent qu'il y a peu de développement qui a été effectué à cet égard. La seule option fournie par Docker pour limiter la quantité d'opérations entrées-sorties à un conteneur ne donne pas un résultat satisfaisant. Les résultats obtenus sur deux conteneurs configurés avec des valeurs de limitation différentes sont en effet presque identiques. Néanmoins, il a été constaté que les options «volumes de données» et «conteneur de volumes de données» quant à elles sont des très bonnes alternatives pour préserver les données des conteneurs Docker.

D'une manière générale, on peut conclure que Docker permet de bien gérer les ressources mémoire, processeur et disque dans ses conteneurs, tout en manipulant les groupes de contrôle et les espaces de noms, quoiqu'il y ait encore pas mal d'améliorations à apporter, surtout en ce qui concerne à la limitation de la bande passante des opérations entrées-sorties de disque dur comme on vient de le mentionner. En outre, Docker ne propose pour l'instant aucune solution pour la gestion de la bande passante réseau, qui comme tout autre ressource requiert un contrôle rigoureux de la part des personnels TI.

ANNEXE I

CONFIGURATION ET INSTALLATION D'UNE MACHINE VIRTUELLE UBUNTU

1. Configuration de la Machine Virtuelle

- Télécharger et installer Oracle VirtualBox

  L'exécutable Oracle VirtualBox peut être téléchargé à partir du lien ci-dessous:

  https://www.virtualbox.org/wiki/Downloads

- Créer une machine virtuelle avec les caractéristiques suivantes :

  Espace disque : 30GB

  Mémoire vive : 2GB

  Mémoire swap : 2GB

  Nombre de CPUs : 2

2. Installation du système d'exploitation Ubuntu sur la VM fraichement créée

- Télécharger Ubuntu à l'adresse suivante :

  http://www.ubuntu.com/download/desktop

- Démarrer la VM après avoir monté l'iso Ubuntu téléchargé

  Suivre les instructions du lien ci-dessous pour l'installation de Ubuntu :

  http://doc.ubuntu-fr.org/tutoriel/installer_ubuntu_avec_le_live_cd

- Mettre Ubuntu à jour après l'installation

  ```
  $ sudo apt-get update
  ```

  ```
  $ sudo apt-get upgrade
  ```

ANNEXE II

INSTALLATION DE DOCKER

1. Installation de Docker

- Installer Docker 1.5 à l'aide de la ligne de commande suivante :

  $sudo apt-get -y install docker.io

2. Activer le module groupe de contrôle (cgroups)

Activer cgroups permet de manipuler les ressources des conteneurs. Sinon on aura l'erreur suivante :

WARNING: Your kernel does not support cgroup swap limit. WARNING: Your kernel does not support swap limit capabilities. Limitation discarded.

- Modifier le fichier /,etc./default/grub pour ajouter la ligne suivante :
  GRUB_CMDLINE_LINUX="cgroup_enable=memory swapaccount=1"

```
GRUB_DEFAULT=0
GRUB_HIDDEN_TIMEOUT=0
GRUB_HIDDEN_TIMEOUT_QUIET=true
GRUB_TIMEOUT=10
GRUB_DISTRIBUTOR=`lsb_release -i -s 2> /dev/null || echo Debian`
GRUB_CMDLINE_LINUX_DEFAULT="quiet splash"
GRUB_CMDLINE_LINUX="cgroup_enable=memory swapaccount=1"
##GRUB_CMDLINE_LINUX=""
```

Figure-A II-1 Extrait du fichier grub du système d'exploitation Ubuntu avec le module cgroups activé

- Mettre à jour noyau (grub) à jour puis redémarrer le système d'exploitation

  $sudo update-grub

  $sudo shutdown –r now

ANNEXE III

MISE À JOUR DE DOCKER

La mise à jour de Docker à la version 1.7 permettra l'utilisation de plusieurs options non disponibles dans la version 1.5 telles que : **--memory-swap -1**, **--cpu-period** et **--cpu-quota** et **--oom-kill-disable**

Mettre à jour Docker

- Vérifier le statut du service docker

  $sudo systemctl status docker

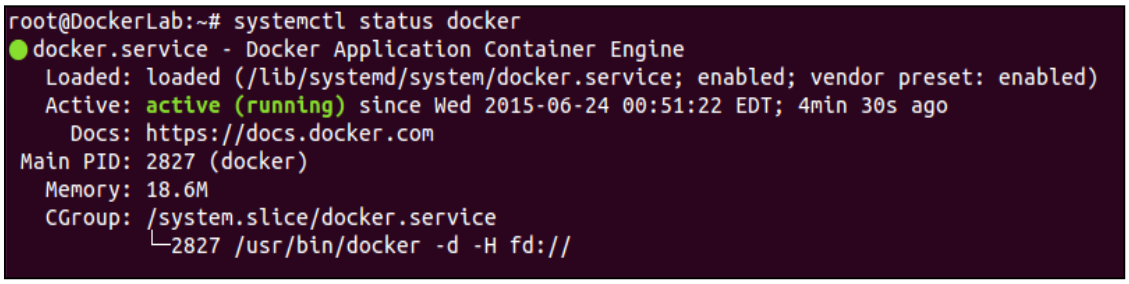

Figure-A III-1 Copie d'écran montrant le service Docker dans un état active

- Arrêter Docker si le service est en marche

  $systemctl stop docker

- Mise à jour de Docker à la dernière version disponible

  $curl -sSL https://test.docker.com/ubuntu | sh

- Démarrer Docker

  Après la mise à jour, quand on essaye de démarrer Docker ($sudo systemctl start docker) on peut avoir les erreurs suivantes :

FATA[0000] Get http:///var/run/docker.sock/v1.18/info: dial unix /var/run/docker.sock: no such file or directory. Are you trying to connect to a TLS-enabled daemon without TLS?
Failed to start docker.service: Unit docker.service is masked.

Pour démarrer Docker avec succès il faut démasquer les services « docker.service » et « docker.socket" en utilisant la commande « unmask »

$sudo systemctl unmask docker.service
$sudo systemctl unmask docker.socket
$systemctl start docker

```
root@DockerLab:~# docker version
Client version: 1.7.0
Client API version: 1.19
Go version (client): go1.4.2
Git commit (client): 0baf609
OS/Arch (client): linux/amd64
Server version: 1.7.0
Server API version: 1.19
Go version (server): go1.4.2
Git commit (server): 0baf609
OS/Arch (server): linux/amd64
root@DockerLab:~#
```

Figure-A III-2 Copie d'écran montrant le résultat de la commande « docker version » qui donne les détails sur la version de Docker installée

ANNEXE IV

UTILISATION DE L'OUTIL « stress-ng »

1. Créer un conteneur

- Créer un conteneur Docker

```
#docker run -ti --name debian-io1 -h debian --cpuset-cpus=1 -m 396m\
 --memory-swap 738m --blkio-weight=500 debian2:evins /bin/bash
```

Caractérisques du conteneur debian-io1:

CPU : cpu ID 1

Mémoire vive : 396 MB

Mémoire swap : 342 MB

- Mettre à jour les packages à l'intérieur du conteneur

```
root@debian-io1:~# sudo apt-get update
```

2. Installer « stress-ng » dans le conteneur

Installer l'outil « stress-ng» en utilisant la ligne de commande ci-dessous:

```
root@debian-io1:~# apt-get install stress-ng
```

3. Faire un essai en appliquant un du trafic sur la mémoire et le processeur

```
root@debian-io1:~# stress-ng --cpu 4 --io 2 --vm 1 --vm-bytes 396M\
 --timeout 360s
```

Utiliser l'outil «htop» pour visualiser l'utilisation des ressources

```
root@DockerLab:~# htop
```

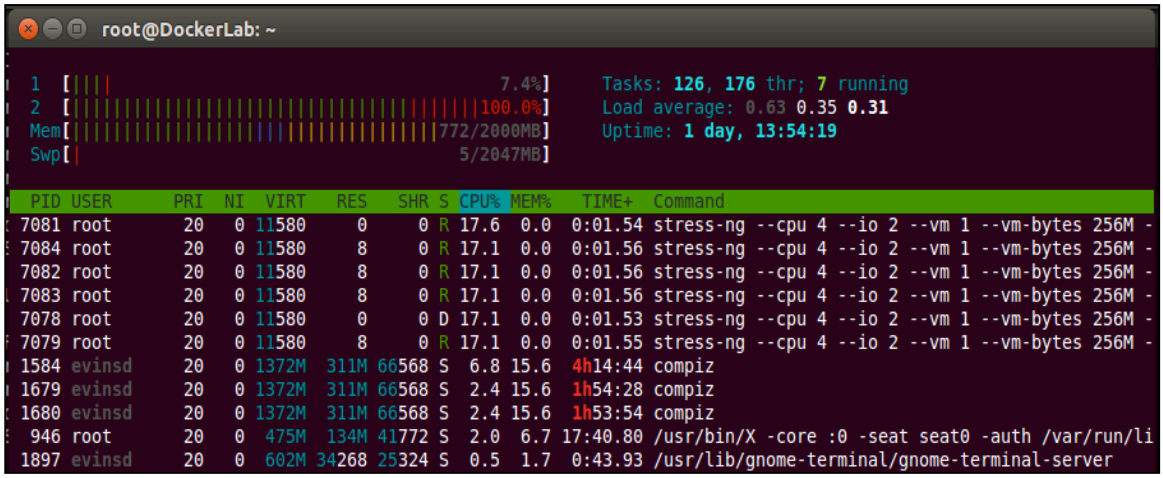

Figure-A IV-1 Capture d'écran montrant le pourcentage d'utilisation du processeur et de la mémoire

LISTE DE RÉFÉRENCES BIBLIOGRAPHIQUES

Dua, R., Raja, A.R., Kakadia, D. 2014. Virtualization vs Containerization to Support PaaS, *Cloud Engineering (IC2E), 2014 IEEE International Conference*, doe: 10.1109/IC2E.2014.41.

Wang, J., Niphadkar, S., Stavrou, A., Ghosh, A.K. 2010. A Virtualization Architecture for In-Depth Kernel Isolation, *43rd Hawaii International Conference on System Sciences (HICSS),* doi: 10.1109/HICSS.2010.41.

Xavier, M.G., Neves, M.V.,et coll. 2013. Performance Evaluation of Container-Based Virtualization for High Performance Computing Environments, *Conference on Parallel, Distributed and Network-Based Processing (PDP), 21st Euromicro International*, doi: 10.1109/PDP.2013.41.

Asay, M. 2013. IDC: Virtualization's March To Cloud Threatens VMware. En ligne. <http://readwrite.com/2013/05/02/idc-virtualizations-march-to-cloud-threatens-vmware>. Consulté le 10 juin 2015.

ISACA. 2010. Virtualization: Benefits and Challenges

Docker Inc. 2015. En ligne. <www.docker.com>. Consulté le 9 Juin 2015»

Chen, Yao, Cai, Wei-Zheng et Zhang, Yu. 2010 The research and implementation of automatic unit test recording framework, *Software Technology and Engineering (ICSTE), 2010 2nd International Conference on*, doi: 10.1109/ICSTE.2010.5608779

VMware Inc. 2010. Guide de gestion des ressources de vSphere, p. 1-10

Felter, Wes, Ferreira, Alexandre, Rajamony , Ram et Rubio, Juan . 2014. An Updated Performance Comparison of Virtual Machines and Linux Containers, "IBM Research, Austin, TX"

Krakowiak, Sacha. 2014. Les débuts d'une approche scientifique des systèmes d'exploitation

Kibe, S., Watanabe, S., Kunishima, K., Adachi, R., Yamagiwa, M., et Uehara, M. 2013. PaaS on IaaS, Advanced *Information Networking and Applications (AINA), 2013 IEEE 27th International Conference.* doi: 10.1109/AINA.2013.73

Xavier, M.G., De Oliveira, I.C., Rossi, F.D., Dos Passos, R.D., Matteussi, K.J. et De Rose, C.A.F. 2015. A Performance Isolation Analysis of Disk-Intensive Workloads on Container-Based Clouds, *Parallel, Distributed and Network-Based Processing (PDP), 2015 23rd Euromicro International Conference*, doi: 10.1109/PDP.2015.67

Polaert , Guillaume . 2014. Docker véritable rupture technologique. En ligne. <http://www.journaldunet.com/solutions/expert/58819/docker-veritable-rupture-technologique.shtml >. Consulté le 10 Juillet 2015

Deka, Ganesh . 2014. Cost-Benefit Analysis of Datacenter Consolidation Using Virtualization, Ministry of Labor & Employment, India

Konrad, Alex. 2015. To Lead The Software Container Rush, Docker Plays Nice With Its Startup Friends. En ligne. <http://www.forbes.com/sites/alexkonrad/2015/07/02/docker-and-its-startup-friends/>. Consulté le 9 Juillet 2015

Salzman, Burian et Pomerantz. 2007. The Linux Kernel Module Programming Guide

Joy, A.M. 2015. Performance comparison between Linux containers and virtual machines, in *Computer Engineering and Applications (ICACEA), 2015 International Conference on Advances in* , doi: 10.1109/ICACEA.2015.7164727

Adufu, Theodora, Choi, Jieun, Kim, Yoonhee. 2015. Is container-based technology a winner for high performance scientific applications?, in *Network Operations and Management Symposium (APNOMS), 2015 17th Asia-Pacific* .doi: 10.1109/APNOMS.2015.7275379

Stubbs, J., Moreira, W., Dooley, R.. 2015. Distributed Systems of Microservices Using Docker and Serfnode, in *Science Gateways (IWSG), 2015 7th International Workshop,*doi: 10.1109/IWSG.2015.16

Guedes, E.A.C., Silva, L.E.T., Maciel, P.R.M.2014. Performability analysis of I/O bound application on container-based server virtualization cluster, in *Computers and Communication (ISCC), 2014 IEEE Symposium*, doi: 10.1109/ISCC.2014.6912556

Beserra, D., Moreno, E.D., Takako Endo, P., Barreto, J., Sadok, D., Fernandes, S. 2015. Performance Analysis of LXC for HPC Environments, in *Complex, Intelligent, and Software Intensive Systems (CISIS), 2015 Ninth International Conference,* doi: 10.1109/CISIS.2015.53

Wubin Li, Kanso, A. 2015.Comparing Containers versus Virtual Machines for Achieving High Availability, in *Cloud Engineering (IC2E), 2015 IEEE International Conference*, doi: 10.1109/IC2E.2015.79

Red Hat, Inc. 2015. En ligne. < https://access.redhat.com/documentation/en-US/Red_Hat_Enterprise_Linux/6/html/Resource_Management_Guide/ch01.html>. Consulté le 10 Juillet 2015.

Linux Kernel Foundation. 2015. En ligne. < https://www.kernel.org/doc/ >, consulté le 11 Juillet 2015.

Babu, S.A., Hareesh, M.J., Martin, J.P., Cherian, S., Sastri, Y. 2014.System Performance Evaluation of Para Virtualization, Container Virtualization, and Full Virtualization Using Xen, OpenVZ, and XenServer, in *Advances in Computing and Communications (ICACC), 2014 Fourth International Conference,* doi: 10.1109/ICACC.2014.66

Carabas, M., Mogosanu, L., Deaconescu, R., Gheorghe, L., Tapus, N. 2014.Lightweight Display Virtualization For Mobile Devices, in *Secure Internet of Things (SIoT), 2014 International Workshop,* doi: 10.1109/SIoT.2014.9

Ning, F., Weng, C., Luo, Y. 2013. Virtualization I/O optimization based on shared memory, in *Big Data, 2013 IEEE International Conference on* , doi: 10.1109/BigData.2013.6691700

Chawla, P., Singh, M., Deep, V., Matharu, G.S. 2015.Systematic overview of mobile virtualization platforms: Comparative analysis, in *Electrical, Computer and Communication Technologies (ICECCT), 2015 IEEE International Conference*, doi: 10.1109/ICECCT.2015.7226063

ElSayed, A., Abdelbaki, N. 2013. Performance evaluation and comparison of the top market virtualization hypervisors," in *Computer Engineering & Systems (ICCES), 2013 8th International Conference*, doi: 10.1109/ICCES.2013.6707169

Yamini, B., Selvi, D.V. 2010. Cloud virtualization: A potential way to reduce global warming, in *Recent Advances in Space Technology Services and Climate Change (RSTSCC), 2010* ,doi: 10.1109/RSTSCC.2010.5712798