Rapport Concepts et Techniques de Virtualisation

Emmanuel FERRANDI - Franck BOURZAT

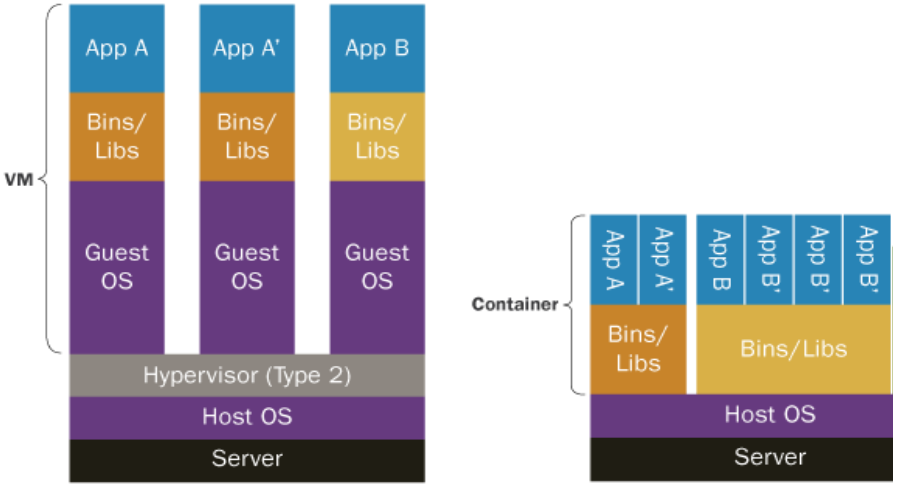
TP2 : tiny.cc/TP_Cloud2
TP1 : tiny.cc/TP_Cloud

Introduction

Ce rapport se compose de deux parties, la première portant sur les conteneurs et les machines virtuelles de façon théorique. On y démontre les avantages et les inconvénients de chacun tout en détaillant leur fonctionnement. Ensuite, vient l'approche pratique, où l'on justifie les dires de la première partie.
En effet, les conteneurs sont un meilleur choix lorsque la plus grande priorité est de maximiser le nombre d'applications à exécuter sur un nombre minimal de serveurs. Contrairement aux machines virtuelles qui sont un meilleur choix pour exécuter des applications qui nécessitent toutes les ressources et fonctionnalités du système d'exploitation lorsque vous devez exécuter plusieurs applications sur des serveurs ou que vous avez une grande variété de systèmes d'exploitation à gérer.

Partie Théorique

Différence entre les 2 grands types de container de virtualisation (VM et CT)



Dans le schéma ci-dessus, on peut voir la présence d'un hyperviseur au niveau de la VM qui permet de la lancer et l'initier. Si l'on regarde la définition, un hyperviseur est une couche logiciell qui permet d'allouer les ressources physiques aux machines virtuelles. On peut ainsi avoir plusieurs VM avec un seul hyperviseur.
On remarque une autre différence entre la VM et le container qui correspond au "Guest OS". Cette partie est en faite le système d'exploitation de la VM, ainsi si on veut mettre Windows sous VM, la partie graphique, réseau, gestionnaire de fichier et tous les composants nécessaires à ce système d'exploitation vont se retrouver ici.
Ensuite les partie identique à la base sont Host OS et Server qui correspondent à la partie Hardware et à l'OS sur lequel on va installer nos VM et containers. Pour la partie haute Bins/Libs et App, on peut prendre l'exemple d'une application Python où l'on va avoir besoin des librairie pour lancer l'application que l'on désire.

Développeur d'application

	VM	Container
Coût de virtualisation (RAM, ROM, CPU…)	La VM va avoir besoin des ressources nécessaires, si on prend le logiciel VMware, on doit avoir 4 Go de RAM, 1 CPU de 4 cœurs cadencés à 2,3 GHz, 32 Go de ROM.	Quant au container, on doit avoir 2.00 GB de RAM, 3.00 GB de ROM.

Utilisation des ressources	La VM comme vu au dessus, a besoin de plus de ressource pour continuer de fonctionner car il est nécessaire de faire tourner un OS en plus. Ces ressources ne seront donc pas utilisées pour l'application du développeur.	Le container va avoir une consommation de ressources bien moindre ce qui va permettre de donner toute la puissance du server/ordinateur à l'application du développeur.
Sécurité	La sécurité se fait au niveau de la machine virtuel, ainsi si cette dernière est sécurisé, le risque est faible de subir une attaque.	Il y a aucune sécurité à l'installation d'un conteneur à la première étape, il est nécessaire de faire des manipulations complexes et longues pour ça. Même si cela est fait, il y a des risques que d'autres conteneurs soient malveillants et aient accés à l'Host OS, pouvant ainsi avoir accès au conteneur de l'application du développeur.
Performance (temps de réponse)	Le développeur peut rapidement et facilement disposer des environnements de développement sans demander le déploiement de VM à son prestataire ou son équipe d'infrastructures. Il est plus autonome.	
Outillage	VMware - VirtualBox - Xen - HyperV - KVM...	Dockers - Linux Lxc...

Administrateur d'infrastructure

	VM	Container
Coût de virtualisation (RAM, ROM, CPU…)	Les machines virtuelles sont un meilleur choix pour exécuter des applications qui nécessitent toutes les ressources et fonctionnalités du système d'exploitation lorsque vous devez exécuter plusieurs applications sur des serveurs ou que vous avez une grande variété de systèmes d'exploitation à gérer.	Les conteneurs sont un meilleur choix lorsque votre plus grande priorité est de maximiser le nombre d'applications exécutées sur un nombre minimal de serveurs.
Utilisation des ressources	L'administrateur	Il est aussi possible de

	d'infrastructure va devoir fournir des serveurs qui pourront supporter la charge des différentes VM ainsi que l'envie des utilisateurs. Ou l'autre possibilité est de limiter l'utilisation pour chaque VM grâce à des outils (listé ci-dessous).	limiter l'usage mais lors de la création du conteneur, mais après l'installation, il est impossible de faire ces modifications. Rendant la tache plus dur pour l'administrateur d'infrastructure.
Sécurité	Il ne va avoir à s'occuper que de la sécurité de l'Host OS et de veiller à avoir des VM avec une sécurité suffisante en fonction de l'application	La sécurité pour les conteneurs va devoir être fait pour chacun d'entre eux, ainsi que de l'Host OS.
Performance (temps de réponse)	Au niveau de l'installation, la VM prendra plus de temps à set-up dû à l'OS à installer. De même pour lancement, où l'OS doit boot tous ses programmes.	Pour l'installation, quelques lignes de commande permettent de mettre en place tous les outils nécessaires à la création d'un ou plusieurs conteneurs. Pour le lancement, il se fait au démarrage et est "équivalent" à une simple application pour l'OS.
Outillage	SolarWinds - PRTG - OpManager - VEEAM - ApexSQL	Kubernetes - OpenStack

Différences entre les types de CT

Il existe différentes technologies de CT sur le marché (e.g. LXC/LXD, Docker, Rocket, OpenVZ, runC, containerd, systemd-nspawn). Leur positionnement n'est pas évident, mais des analyses comparatives sont cependant disponibles sur le Web, souvent par le bais de figure telle que la Figure 2 ci-dessous.

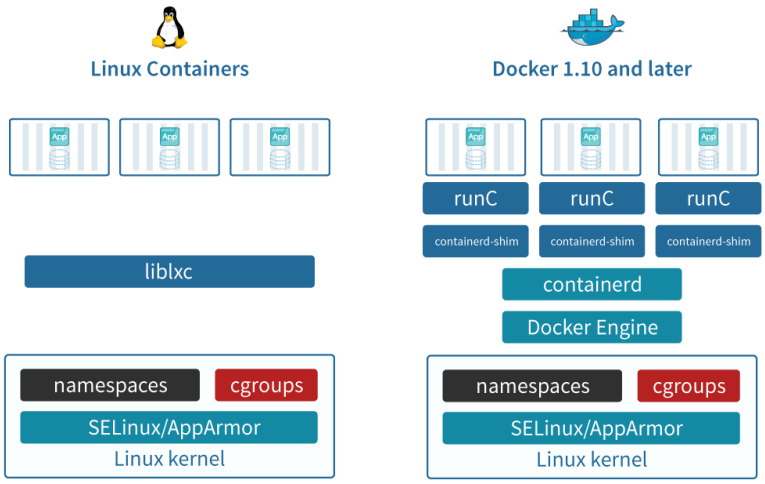


Figure 2 : Linux Lxc vs Docker

Ces technologies peuvent toutefois être comparées à partir de critères tels que les suivants (liste non exhaustive) :

- Isolation des applications / ressources (au sens “multi-tenancy application”),
- Degré de conteneurisation (e.g. système, application),
- Outillage (e.g. API, outils pour l’intégration continue ou pour la composition de services).

L’isolation correspond au niveau matériel (proc, mémoire) au recours ou non à des machines virtuelles à part entière.
Le degrés de conteneurisation correspond au niveau à partir duquel on applique la conteneurisation.
L’outillage correspond aux outils logiciels sur lesquels on peut s’appuyer derrière ces technologies.

Nous présentons cela dans le tableau suivant :

	Isolation	Degré de conteneurisation	Outillage
Docker/ Rocket	Pas d’isolation	Docker n'inclut pas de système d'exploitation, mais s'appuie au contraire sur les fonctionnalités du système d’exploitation fournies par la machine hôte. Ils vont virtualiser des micro services (processus et applications) et non un système d'exploitation.	Clint Docker, API de haut niveau fournissant une solution pratique de virtualisation (Docker Engine API), SDK for Python and Go intégration continue possible, composition de services au sein d’un conteneur dédié
LXD	Isolation totale	Prise en charge fondamentale de la	Commandes comme

	des ressources	conteneurisation du système d'exploitation	*lxc-create*/destroy avec le command line tool, intégration avec OpenStack, OpenNebula

Ce tableau ne différencie pas Docker de Rocket sur les critères du tableau ci-dessus cependant au niveau sécurité Rocket est dit moins vulnérable car il ne permet pas la création de nouveaux container en mode root.

Pour les autres critères on peut citer la persistance des données : pour docker elle est possible mais n'est pas faites pour. LXD quant à lui à son propre environnement donc la persistance est assurée.

Différence entre les 2 grands types d'hyperviseurs (Type 1 et 2)

En théorie, il existe 2 grandes familles d'hyperviseurs dites de type 1 ou de type 2.

Un **hyperviseur de type 1** a comme particularité de s'installer directement sur la couche matériel (à comprendre qu'il est relié directement au matériel de la machine hôte). Il est alors considéré comme outil de contrôle du système d'exploitation, c'est à dire qu'il s'agit d'un noyau allégé et optimisé pour la virtualisation de machines, à contrario d'un OS classique (Windows ou Linux). Toutefois, il est possible d'exécuter uniquement un hyperviseur à la fois sur un serveur.

Un **hyperviseur de type 2** est installé sur l'OS ce qui permet d'en avoir plusieurs sur la même machine à contrario du type 1. Mais cette contrepartie demande des ressources supplémentaires pour faire tourner l'Host OS ce qui en laisse moins pour les OS au dessus. Cette architecture est utilisée par les logiciels suivants : VMware Player, VMware Workstation, VirtualPC et VirtualBox.

OpenStack utilise KVM (type 2) et Xem (type 1), il est donc possible d'utiliser OpenStack avec les deux configurations suivantes.

Différence entre les 2 principaux modes de connexion au réseau d'un container de virtualisation

Sous certains hyperviseurs, il existe plusieurs possibilités pour connecter une VM / CT à l'Internet, via la machine sur laquelle elle a été créée (ici votre PC, appelé "PC hôte" par la suite). Les 2 principaux modes sont les suivants :

- le mode *NAT* est celui appliqué par défaut : il ne requiert aucune configuration particulière. Suivant ce mode, la VM / CT est connectée à un réseau IP privé (ie. à adresse privée) et dispose d'un routeur virtuel (géré par l'hyperviseur) au sein du PC hôte pour communiquer avec l'extérieur de son réseau :
    - ce routeur met en œuvre l'équivalent d'une fonction de NAT (*postrouting*

seulement) permettant à votre VM de communiquer avec son PC hôte ou avec l'extérieur ;
    - en revanche, la VM n'est pas visible depuis l'extérieur du PC hôte (ou depuis une autre VM hébergée sur le PC hôte) : pour la rendre accessible depuis l'extérieur, il est nécessaire de mettre en œuvre un port forwarding (*prerouting*) au niveau de VirtualBox.
- suivant le mode *Bridge*, le plus courant (mais pas systématique) est que la VM / CT se voit virtuellement connectée au réseau local du PC hôte (cf. Figure 3) : elle possède alors une adresse IP l'identifiant sur le réseau du PC hôte et accède (ou est accédée) à (depuis) l'Internet comme le PC hôte.

NB : vous pourrez également regarder les autres modes disponibles sous VirtualBox, notamment le mode réseau privé.

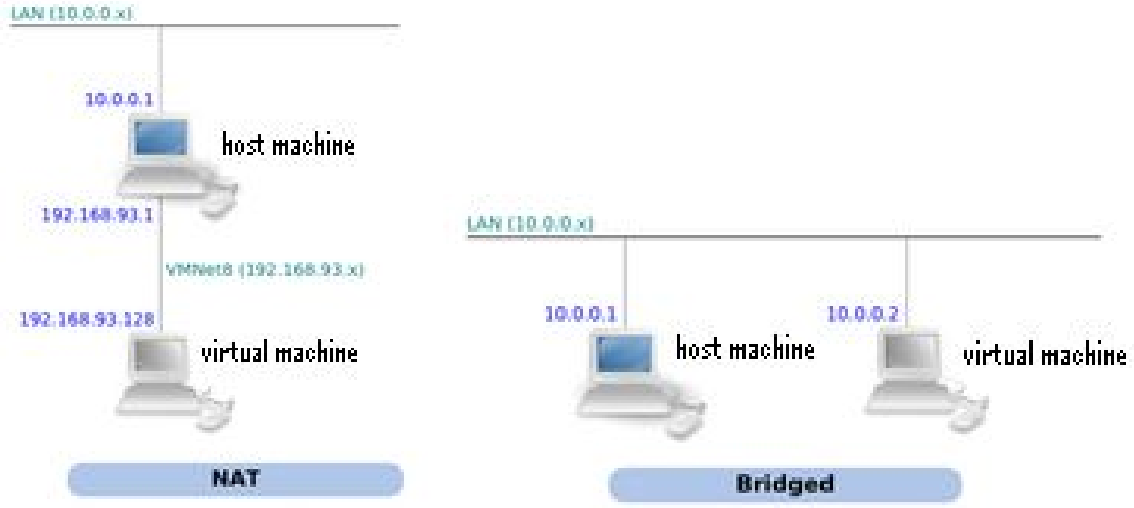


Figure 3 : Mode NAT vs Mode Bridge (version la plus courante)

Partie Pratique

Test de connectivité

Ci-dessous la configuration réseau de notre PC hôte ainsi que notre machine virtuelle.

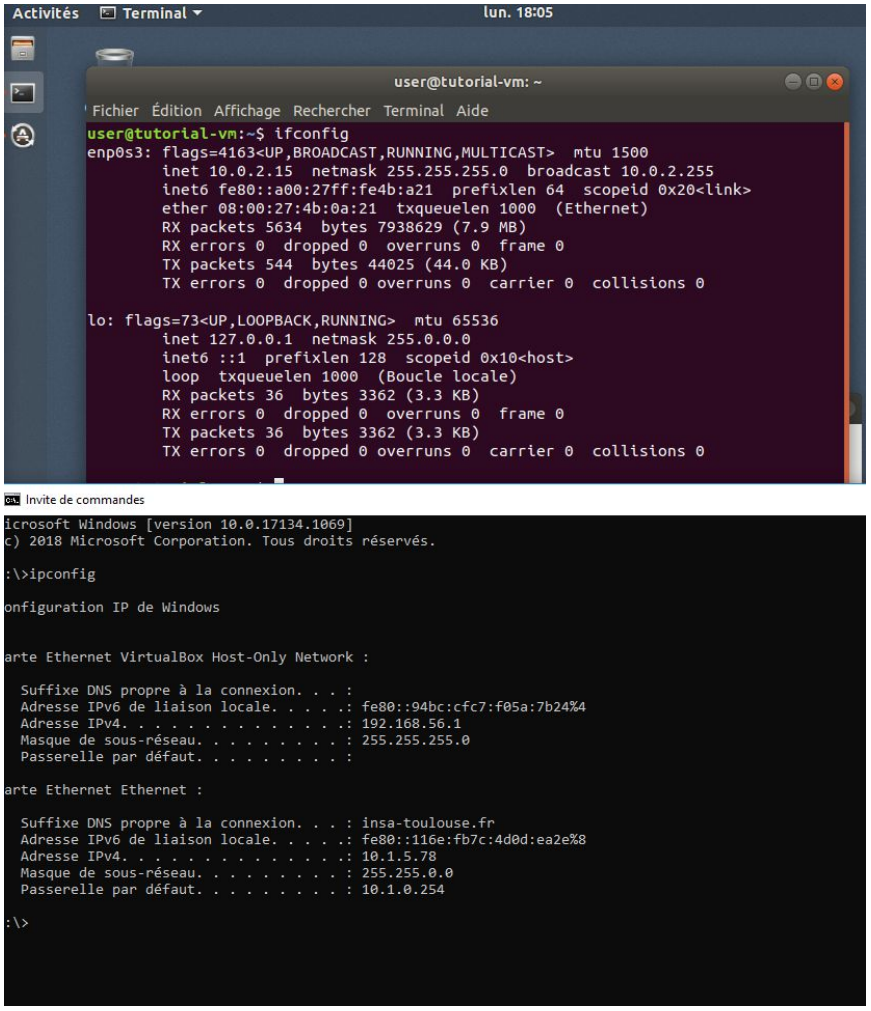


fig 4 - configuration réseau

Le mode NAT étant configuré par défaut, il est impossible de ping de l'extérieur vers la VM car la VM est caché par le NAT. Il faut mettre en place un port forwarding présentée ci-dessous (port 22).

Nom	Protocole	IP hôte	Port hôte	IP invité	Port invité
Rule 1	TCP	10.1.5.86	2000	10.0.2.15	22

Cependant, la VM peut ping le PC hôte de même que les autres ordinateurs de la salle.

5ème partie = Installation et approvisionnement (provisioning) de conteneurs de type Docker sur une VM

La connectivité est résumée dans la figure suivante :
Les flèches indiquent les connectivité possible entre les différentes entités.

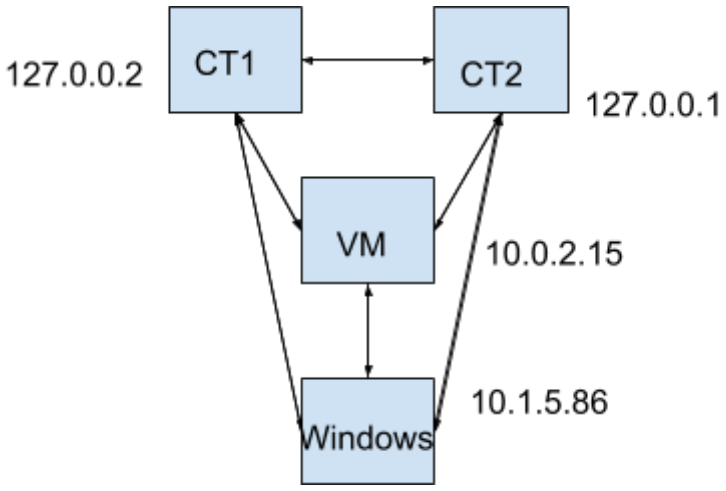


fig 4 - Résumé des connectivités

Par la suite, nous avons créée une nouvelle instance dans le but de créer CT2. Puis, on y installe la commande nano.
Lors de la création de CT3, on copie l'image de CT2 d'où la présence de nano dans ce nouveau conteneur.

Opération sur des VMs avec OpenStack

L'adresse IP affectée par l'hyperviseur à la VM est une adresse privée essayant de se raccorder au réseau public de l'INSA. Or cette configuration échoue, nous devons crée un réseau privé par l'intermédiaire d'un routeur.
(voir figure)

Connectivité

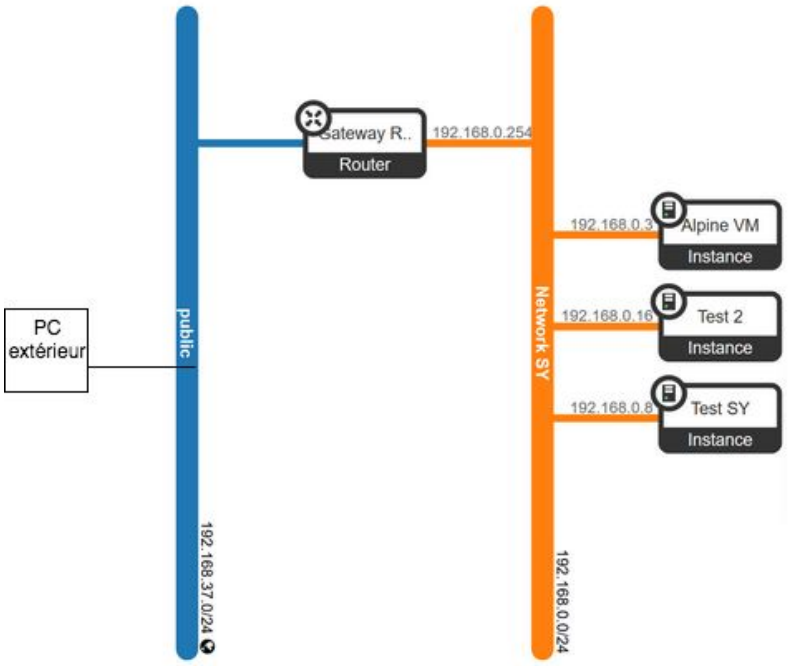
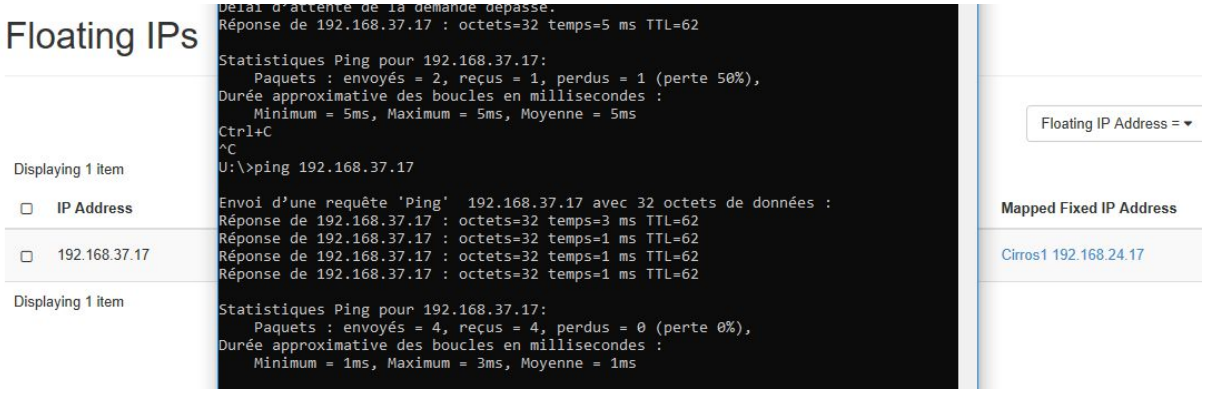


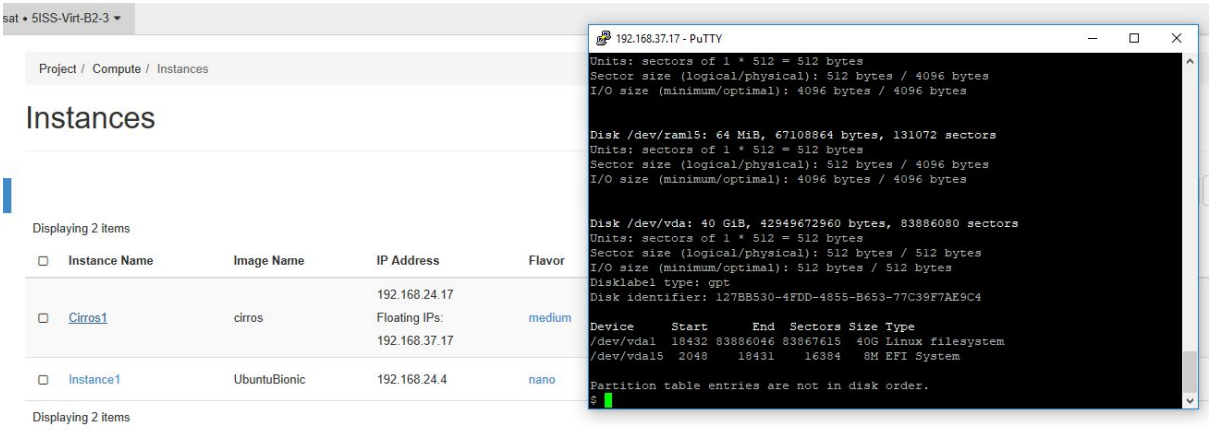
Figure 5 : Target Network Topology

Ici, on peut ping depuis le réseau privé vers l'extérieur mais il est impossible depuis un PC extérieur de ping les machines situées dans le réseau privé par l'intermédière de la gateway. On retrouve ici le même problème que précédemment, il faut passer par le Floating IP où l'on associe une adresse IP à la machine et grâce à un routage optimal de la part d'OpenStack, il crée une adresse IP au niveau du réseau public.
On peut désormais grâce à ce changement de paramètre effectuer des pings et se connecter en SSH sur la VM. A condition évidemment d'avoir autoriser ces types de connexion auparavant sur OpenStack.

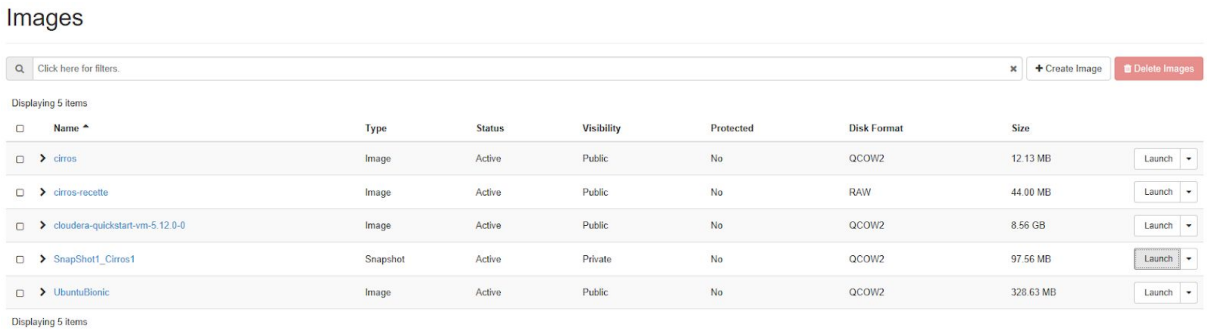


Désormais on souhaite sauvegarder cette VM afin d'en faire une image et la réutiliser dans le futur pour d'autres VM similaire. OpenStack a prévu tout ça grâce à l'outil Snapshoot, qui va créer l'image de la VM souhaitée. Avant de réaliser ça, il est important de redimmensionner la VM afin qu'elle puisse être à la taille que l'on désire pour nos futures

VM. On utilise l'outil Resize mais si on réduit trop l'espace alloué notre VM ne pourra plus fonctionner normalement où avec des lenteurs. Il est important de prendre en compte se paramètre pour éviter toute déception et suppression non désirée.
On resize donc notre machine et lorsque l'on va dans les paramètres disques de cette dernière, on constate bien le changement :



Ainsi notre machine a la taille désirée et on peut désormais sauvegarder son image.



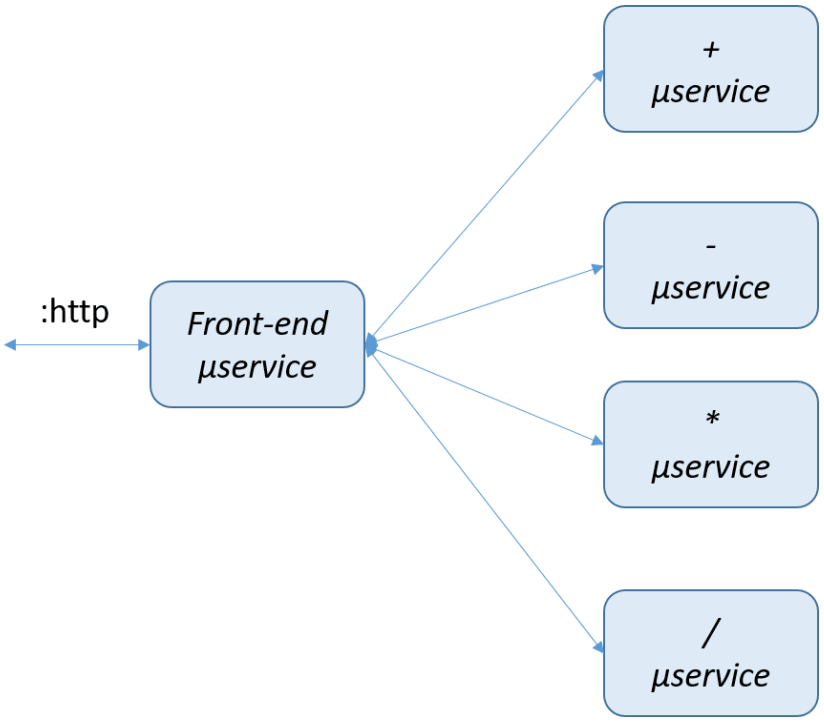
On créée une nouvelle instance à partir de cette image et on constate qu'elle est identique à la précédente.

Calculatrice sur OpenStack avec des VMs

Dans cette partie, on cherche à réaliser une architecture avec plusieurs VMs pour faire une calculatrice. On va avoir besoin de plusieurs outils :

- le front-end
- l'additionneur
- le soustracteur
- le diviseur

On va donc se retrouver avec cette architecture :



Le front-end va permettre de lier tous les autres services. Ainsi on pourra poser notre calcul au front-end qui va se charger de dispatcher les calculs avec leur priorité aux autres sous services.
On créé chaque VMs dans un premier temps, on utilise des images d'Alpine données dans OpenStack. On installe les différents scripts et les dépendances nécessaires, ce qui n'a pas été facile dû à des VMs avec de vieilles versions.
Une autre étape est requise pour set-up le programme JavaScript du front-end pour qu'il ait les bonnes adresses IP des autres micro-services.
On lance donc dans un premier temps le micro-service + et ensuite le Front End. On obtient donc ces différents affichages.

On tape la commande de requête *curl* pour contacter le micro-service CalculatorService:

```
alpine-node:~# curl -d "5+6" -X POST http://192.168.24.9:50000
result = 11

alpine-node:~# _
```

Qui reçoit la commande et détecte une addition et demande de faire l'addition au micro-service lié.

```
alpine-node:~# node CalculatorService.js
Listening on port : 50000
New request :
5+6 = 11
```

Le micro-service additionneur fait son travail et fait le calcul. Il renvoie ensuite la réponse au micro-service qui a demandé.

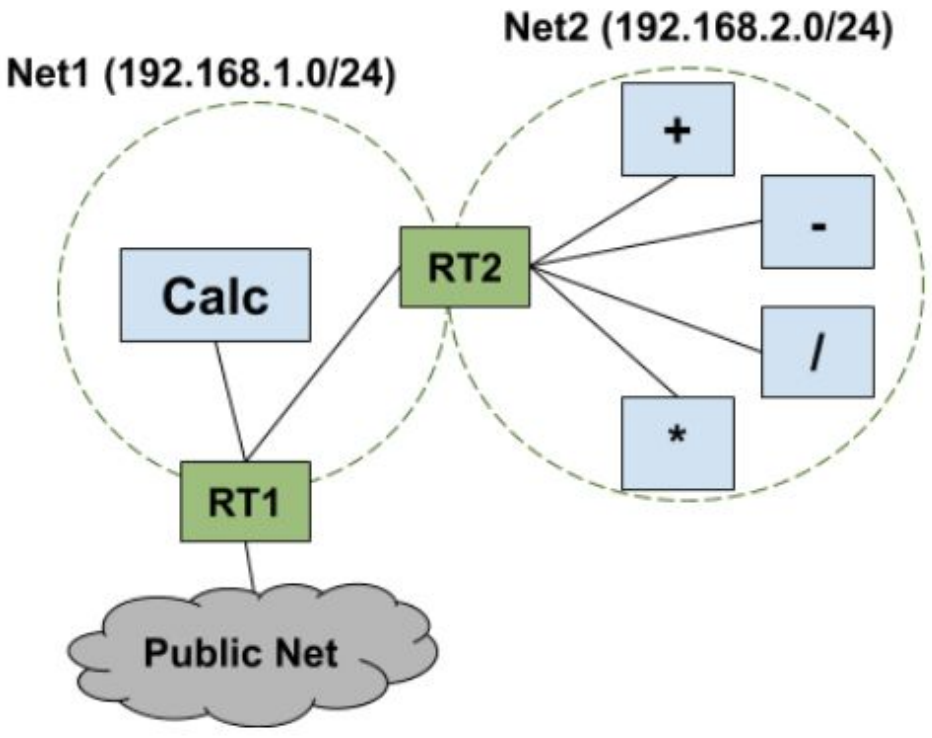
```
New request :
A = 5
B = 6
A + B = 11
```

Ainsi on remonte la chaîne et l'ordinateur qui a exécuté le script va avoir sa réponse au calcul demandé.
Cette exemple basique permet de voir l'étendu des possibilités que peut avoir OpenStack, mais aussi l'intérêt des machines virtuels.
Nous nous sommes posés la question sur l'intérêt réel de faire cette application sur des VMs et non des containeurs qui demandent moins de ressources. Dans notre cas, on a mobilisé 12 Go de RAM et 150 Go de ROM juste pour une calculatrice qui aurait pu prendre seulement quelques Mo avec des containeurs.

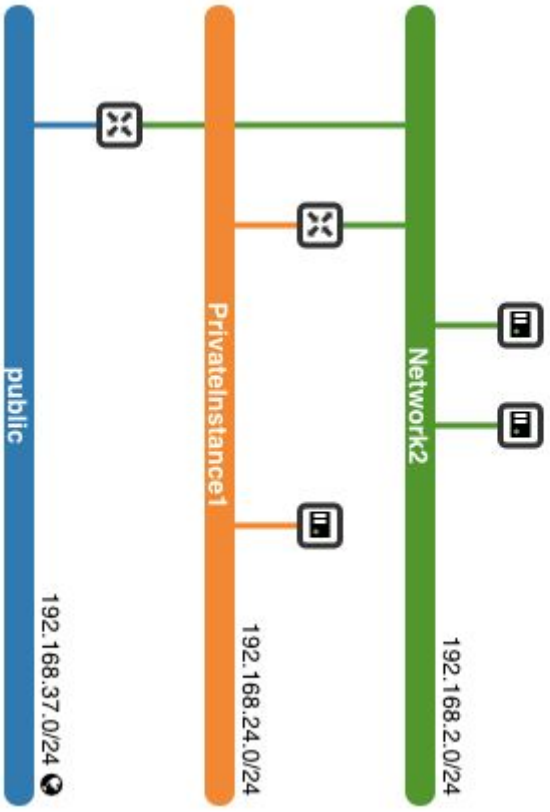
Déploiement de notre calculatrice sur un réseau plus complexe

Dans cette partie, nous allons procéder à une requête éventuelle d'un client et proposer une topologie réseau virtualisé sur demande. Cette topologie inclue des routeurs, des hôtes virtuels et des fonctions de communication qui implémentent une partie de la couche réseau

Création de deux sous-réseau suivant la topologie suivante :



Le réseau privé numéro1 réutilise l'instance créée précédemment. Nous avons juste créé un sous réseau (network2) puis connecté les différents routeurs suivant la topologie suivante et ajouter le port "addition".



Implémentation de la topology ciblé sur openStack

Lors de la création des sous réseau, on a déplacé les machines virtuelles ce qui a supprimé les interfaces ethernet et les routes par défaut. Deux choix s'offrent à nous qui est de faire la configuration à la main ou de recréer les machines. On a décidé de refaire les routes et les interfaces à la main. Cette étape fut longue et fastidieuse mais nous y sommes arrivés.

Lorsque l'on essaie de se connecter à internet avec un des services d'addition ou de soustraction, aucune connexion à internet est possible. En effet, le réseau PrivateInstance ne connaît pas la route pour accèder à internet. Il faut donc ajouter des routes statiques pour permettre ce lien.

Provisionning End-user Application in Cloud Platforms

Dans cette partie, on va découvrir plusieurs type de plateforme de cloud. Elles ont des particularités de modification comme on peut le voir dans le tableau suivant. En effet, si on

déploie notre code grâce à l'Infrastructure en tant que Service, on a la possibilité de modifier le code, les librairies et les JVMs.

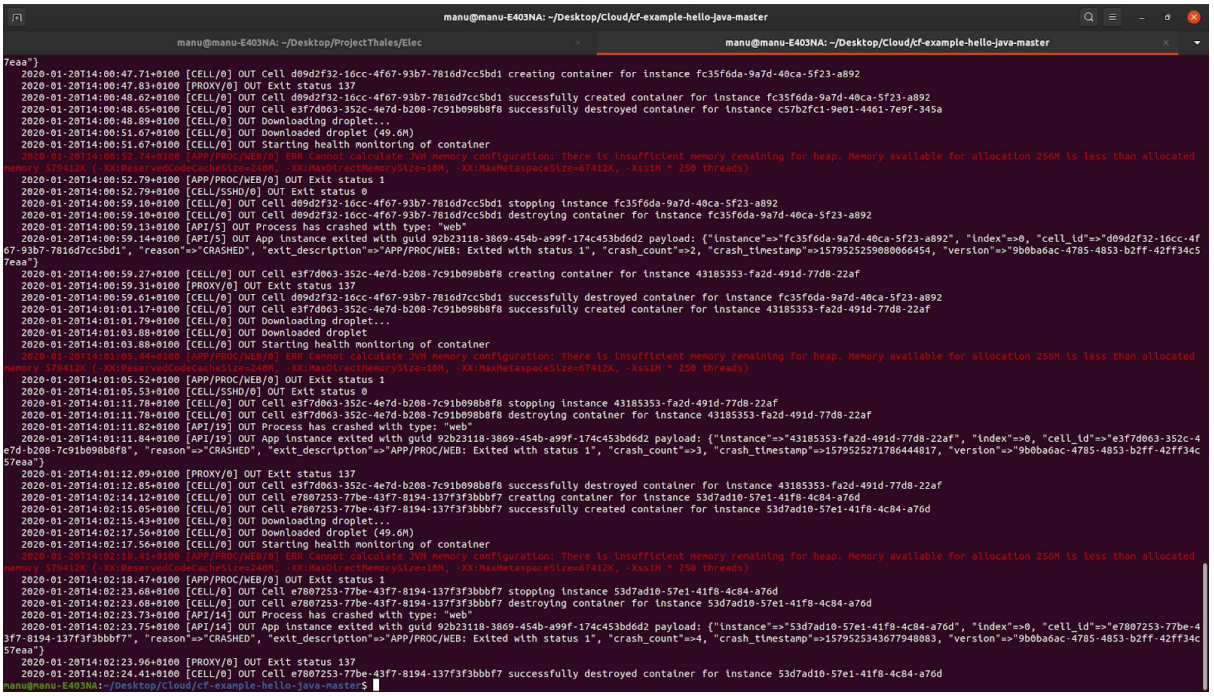
App deploiement	IaaS	PaaS
code	code	code
Lib	Lib	Lib
JVM	JVM	JVM
OS	OS	OS
Physique	Physique	Physique

Google Cloud

Après de nombreuses difficultés, nous n'avons pas réussi à implémenter ce service.

Pivotal Web Services

Après suivi les différentes procédures pour mettre lier notre programme CF CLI à notre compte, on essaie de push notre programme sur Pivotal. On tombe sur une erreur.

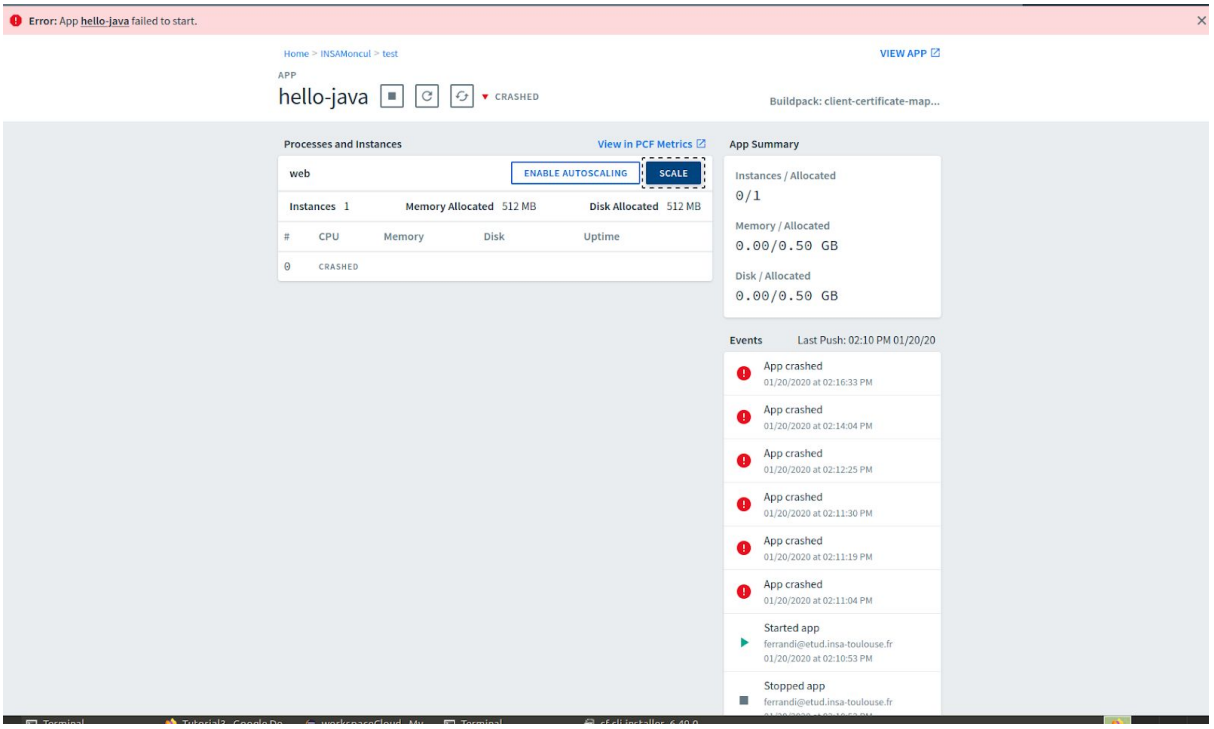


On constate que la mémoire RAM allouée n'est pas suffisante au projet Hello-Word. On doit donc changer cette valeur dans *manifest.yml*.

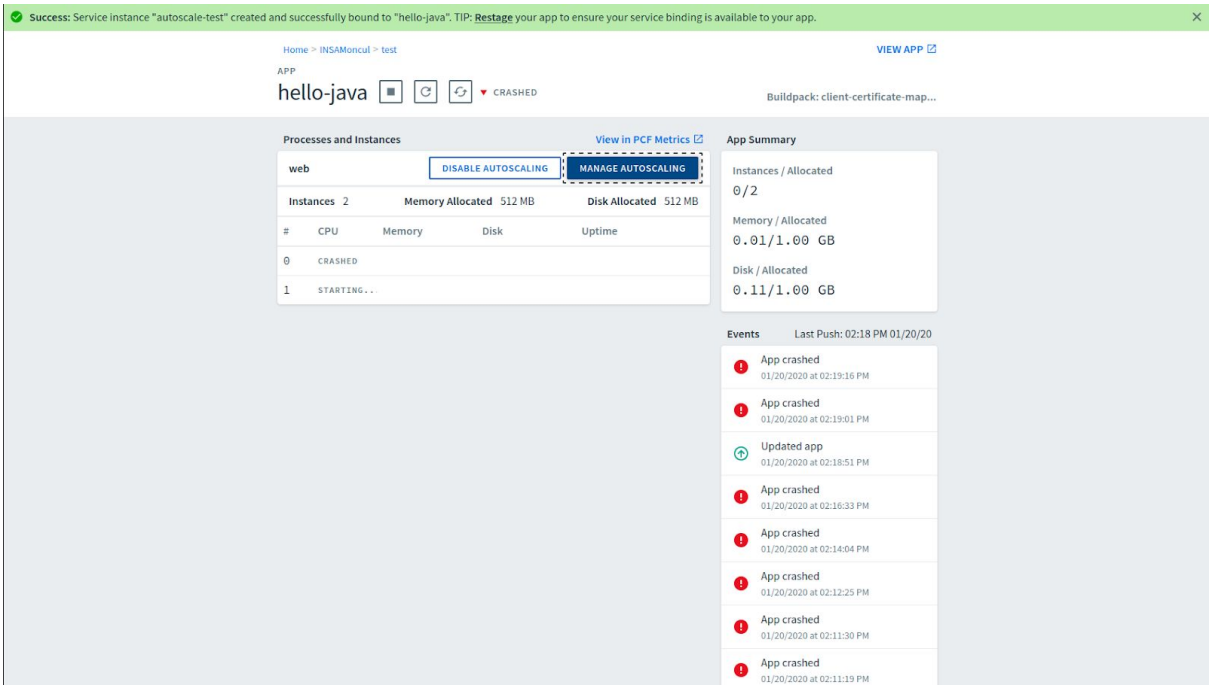
On peut après cette étape aller sur le service créé

```
Java is an island
```

Par la suite on tente de faire un scale Down pour savoir si la ressource va supporter le changement de ressources. Sans grande surprise, lorsque l'on alloue 500Mo de RAM et 500Mo de ROM, l'application ne démarre plus.

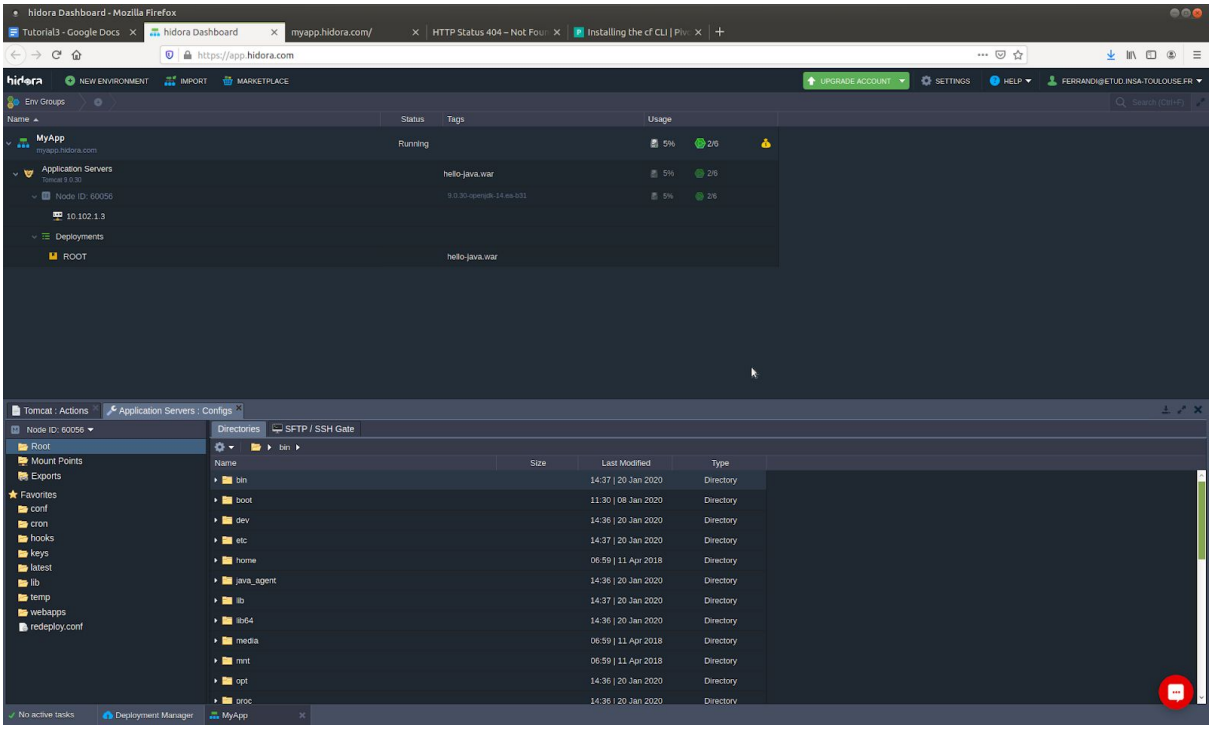


Pour remédier à ce problème on ENABLE AUTOSCALING qui va gérer tout seul les ressources nécessaires à notre application.



Hidora

Dans cette partie, nous allons utiliser une instance publique déployé sur Hidora.
En suivant la procédure, on arrive bien à lancer le programme Java grâce à Tomcat server, comme on le voit sur la photo suivante.



Conclusion

Ces travaux pratiques nous ont permis d'appréhender les technologies du cloud. Dans une première approche théorique nous avons étudié deux grands types de container de virtualisation (VM et CT), différences entre les types de container, types d'hyperviseurs et les principaux modes de connexion au réseau d'un container de virtualisation.
Dans la partie pratique, nous nous sommes penchés dans un premier temps sur les technologies de virtualisation, leurs différences entre celles-ci et les différentes catégories dans lesquelles elles appartiennent (machines virtuelles, conteneurs légers…). Ensuite, nous avons configuré des machines virtuelles à l'aide de VirtualBox puis nous avons vérifié des conjectures en relation avec la connectivité réseau notamment vers l'extérieur. (connection à internet)

Dans la deuxième partie, nous avons cherché à réaliser une architecture avec plusieurs VMs pour faire une calculatrice. Cette exemple basique nous a montré l'étendue des possibilités que peut avoir OpenStack, mais aussi l'intérêt des machines virtuels. Enfin, malgré des problèmes logiciels rencontrés, nous avons entrevue des plateformes d'hébergements cloud utilisés aujourd'hui dans les entreprises comme Hidora ou Cloud Foundry.