INSA INSTITUT NATIONAL DES SCIENCES APPLIQUÉES TOULOUSE

RAPPORT TP CLOUD

Adaptability - Cloud and Autonomic Management

SOUMIS PAR

Yacine BENCHEHIDA 5ISS-A2
Hugo LE BELGUET 5ISS-A2

SOUS LA DIRECTION DU

Prof. **Sami YANGUI**

Année scolaire: 2021-2022
Novembre, 2021

Table des matières

Compte-Rendu TP Cloud

Objectifs 1 à 3

1. Similitudes et différences entre les principaux hôtes de virtualisation (VM et CT)

a) Définitions

i) Machine Virtuelle

Une machine virtuelle (VM) est une ressource informatique qui utilise un logiciel au lieu d'un ordinateur physique pour exécuter des programmes et déployer des applications. C'est un système d'exploitation installé sur un logiciel qui imite un matériel dédié (hardware).

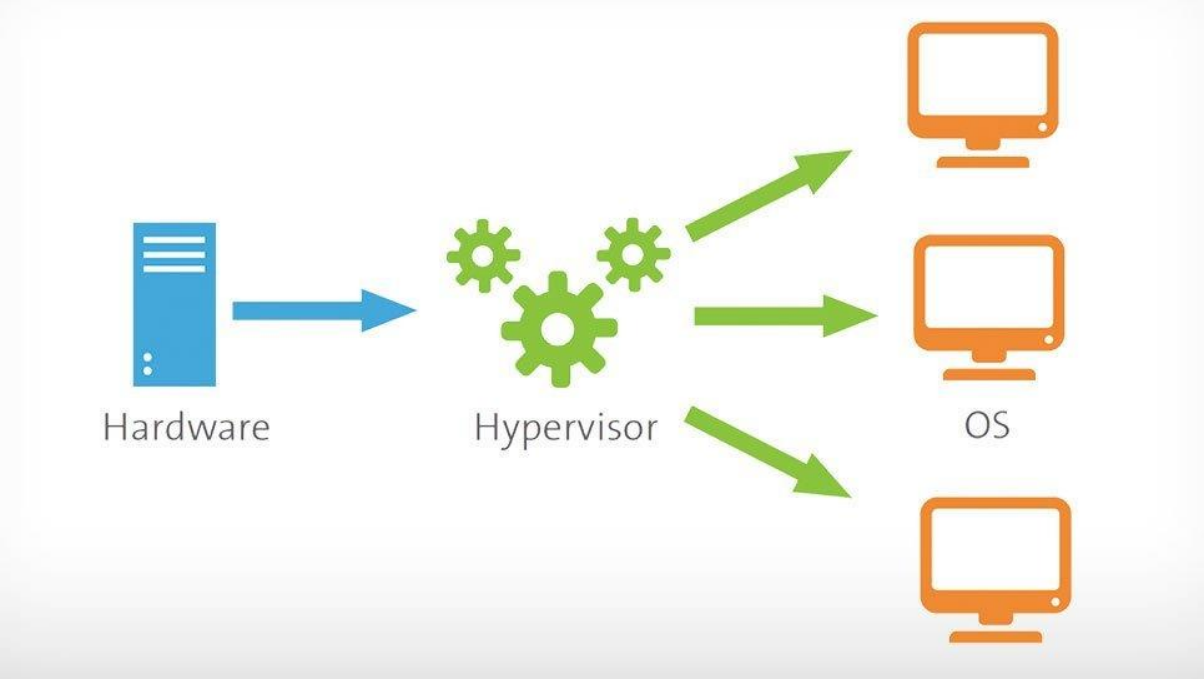


Figure 1 : Fonctionnement de la création de machine virtuelle

L'hyperviseur, ci-dessus dans la figure 1, émule intégralement les différentes ressources matérielles d'un serveur ou PC client, telles que :

- La RAM
- L'espace de stockage
- Le nombre de CPU

- le réseau ...

De même, il peut émuler plusieurs plateformes matérielles virtuelles isolées les unes des autres. Chaque machine virtuelle exécute son propre système d'exploitation et fonctionne séparément des autres VM, même si elles sont toutes exécutées sur le même hôte.

j) Conteneur (Docker)

Docker est un outil qui permet d'embarquer et d'exécuter une application dans un **environnement cloisonné appelé "conteneur"**.

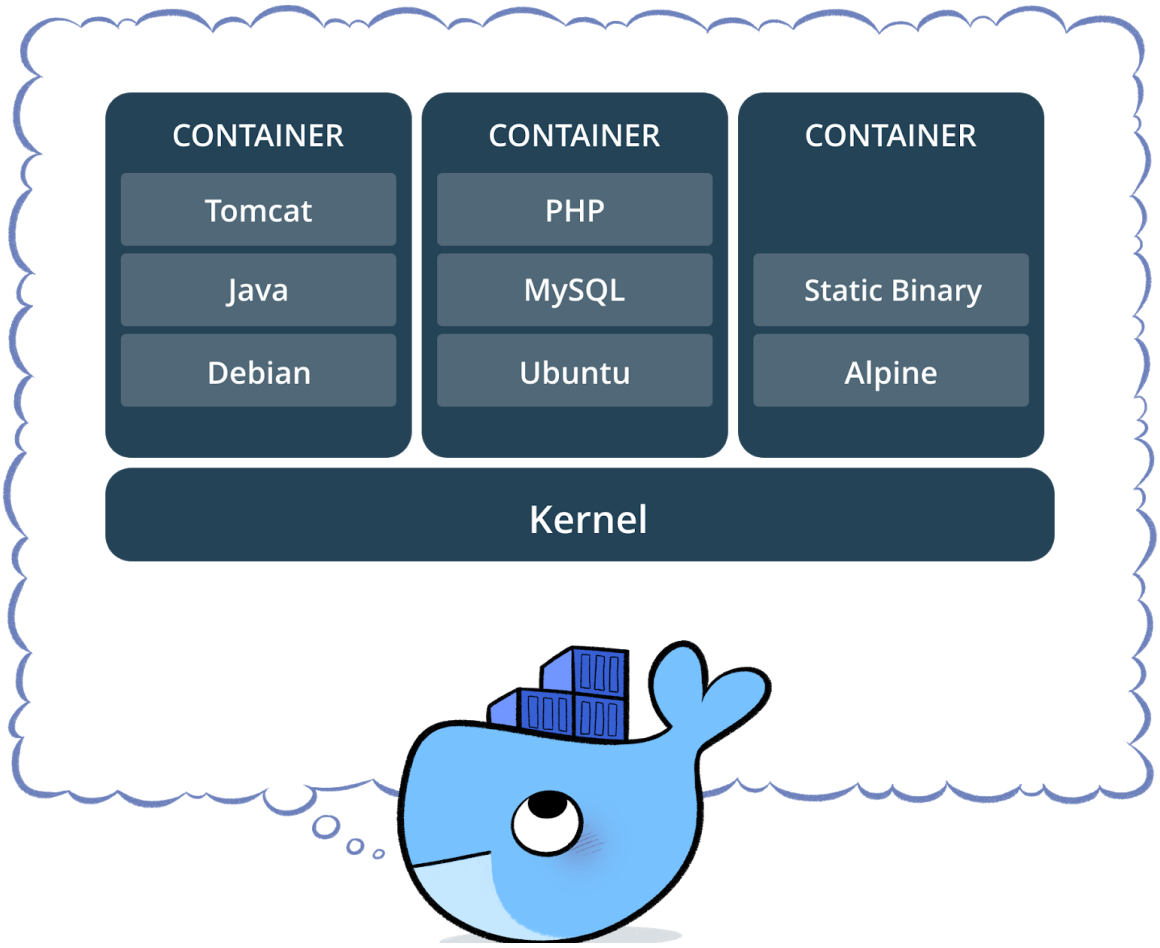


Figure 2 : Différents conteneurs fonctionnant sous le même OS

Le **conteneur** fournit une virtualisation au niveau du système d'exploitation contrairement à une machine virtuelle classique, qui elle fournit une virtualisation au niveau matériel.

Le rôle de l'hyperviseur est alors remplacé par un **moteur de conteneurisation** tel que "Docker", qui lui va créer les conteneurs directement par-dessus l'OS.

b) Comparez les deux types d'hôtes selon deux perspectives : du point de vue d'un développeur d'applications et du point de vue d'un administrateur d'infrastructures

i) Machine Virtuelle vs Conteneur

	VM	Conteneur
virtualization cost, taking into consideration memory size and CPU	L'hyperviseur et le système d'exploitation "virtualisé" ont des tâches d'arrière-plan qui **utilisent la mémoire et le CPU de l'OS hôte**. A cause de cela, les autres applications **auront moins de ressources à leur disposition**. Il faut donc une machine performante (CPU, RAM, …) pour pouvoir faire tourner plusieurs vm en même temps !	Le rôle de l'hyperviseur est alors remplacé par un **moteur de conteneurisation** tel que "Docker", qui lui va créer les conteneurs directement par-dessus l'OS. Le déploiement d'un environnement appelé "micro-service", est donc **beaucoup plus léger** en termes de mémoires, de performances et de CPU.
Usage of CPU, memory and network for a given application	Une VM utilise beaucoup plus de ressources contrairement au conteneur. Elle exécute plusieurs OS distincts qui eux doivent **passer par la couche de virtualisation.** En outre, une VM **utilise beaucoup plus de ressources** en terme de mémoire car elle consomme de la mémoire même si elle n'exécute pas de processus. L'utilisation du CPU n'est pas aussi importante car la virtualisation **n'est pas très exigeante en puissance de calcul.**	Contrairement à une VM classique, **il n'y a pas d'application exécutant d'arrière-plan** dans un conteneur, ce qui a pour effet de largement réduire le coût en mémoire ainsi qu'en utilisation réseau.
Security for the application (access right, resources sharing, etc.)	Les ressources sont allouées pour la VM. Si l'hyperviseur n'a aucune vulnérabilité de sécurité, l'hôte et l'OS invité devraient théoriquement être	Un conteneur est installé par-dessus l'OS. Les ressources sont donc partagées entre l'OS invité et les conteneurs fonctionnant

	maintenus séparément ainsi que toutes les autres VM que par des fichiers et le réseau. Dans ce contexte, la VM est **le meilleur support de virtualisation en termes de sécurité**	sur l'OS. La visibilité sur les ressources utilisées n'est pas aussi simple qu'avec les VM. En termes de sécurité, puisque tous les conteneurs partagent des ressources, si l'un d'entre eux est vulnérable, **il pourrait rendre les autres vulnérables** voir *l'OS hôte.*
Performances (response time)	En termes de performances, VM prend beaucoup plus de temps à se lancer contrairement à un conteneur puisqu' il doit : ● Démarrer un système d'exploitation complet ● Démarrer un processus de démarrage ainsi que tous les autres services ...	En termes de performances, un conteneur est beaucoup plus rapide de par sa légèreté et le fait qu'il n'exécute que les applications de haut niveau. C'est donc très rapide à lancer ou modifier. En effet, cela s'explique par la logique de Docker qui est : un "micro-service" par conteneur.
Tooling for the continuous integration support	Une VM n'est pas dédié à être portable pour plusieurs raisons : ● Différents logiciels libres de virtualisation qui délivrent des VM propres à leur logiciel. C'est-à-dire qu'une VM crée sur VirtualBox ne pourra pas être exporté sur un autre logiciel tel que Proxmox (possible mais complexe) ● La taille de la VM est considérable (plusieurs Go en général)	Un conteneur est le meilleur support portable. Il est conçu pour être exporté et permettre de s'exécuter dans différentes machines. En effet, via Docker, on peut soit utiliser une image conçue dans le Docker Hub, soit créer sa propre image, la télécharger et la charger dans différentes machines via une clé USB ou protocole de transfert de fichier (scp, tftp, sftp …)

j) Développeur vs Administrateur

	VM	Conteneur
Dev	L'utilisation d'une VM ne requiert aucune connaissance en plus par rapport à l'utilisation d'un OS classique. La principale différence vient du partage, si le développeur veut partager son travail effectué sur VM, la taille de l'image et la compatibilité seront deux problèmes majeurs, qui n'existent pas avec un conteneur. De plus, le travail collaboratif n'est pas forcément évident sur VM.	Des notions en Docker sont requises par contre. Grâce à la portabilité d'un conteneur, déployer et partager notre travail est très facile. De même, les conteneurs peuvent être également être utilisés pour rapidement télécharger, exécuter et déployer des applications via le Docker Hub. Il suffit de télécharger simplement dans le Docker Hub, une image adaptée à nos besoins : serveur tftp, base de données Mango…. et l'exécuter en tant que conteneur.
Admin	Pour un admin, une VM permet d'avoir un contrôle total sur les ressources que l'OS virtuel utilise. Bien que le procédé puisse être exigeant en ressource, il s'agit seulement de s'assurer que la VM ne manque jamais par rapport à ce dont elle a besoin. De plus, une VM est visible sur le réseau de la même façon qu'un pc classique et les mêmes règles s'y appliquent. C'est donc beaucoup plus simple à administrer, plus particulier via des outils de monitoring, disponible pour ce type de support.	Du point de vue d'un administrateur, utiliser un conteneur n'est pas la première chose à laquelle il va penser. En effet, il doit gérer une infrastructure, le plus souvent avec un pc de management qui lui s'occupera de la maintenance et la supervision de cette infrastructure. En effet, il a des outils pour superviser son infrastructure (graylog, Nginx, ...) De plus en plus d'entreprises et donc d'administrateurs ont choisi de "dockeriser" ces outils : de les prendre directement sur le Docker Hub (docker graylog, docker ansible …). Même si les librairies/bins sont séparées, un impact sur

		le disque dur d'un conteneur peut affecter un autre conteneur. L'allocation des ressources réseau peut être délicate (pas de mode bridge par exemple). Outils de monitoring disponibles. Donc, même si ce n'est pas la fonction première, Docker devient et va devenir un outil pour un admin.

2. Similitudes et différences entre les types de conteneurs existants

	LXC	Docker	Rocket
Application	Possibilité d'avoir plusieurs applications par conteneur	Fait pour n'avoir qu'une seule application par conteneur	Possibilité d'avoir plusieurs applications par conteneur
Utilité de la communauté	Communauté vivante et active	C'est la communauté la plus utilisée, particulièrement grâce au "docker hub image sharing"	Communauté vivante et active
Sécurité/Isolation	S'exécute directement sur la machine, pas réellement d'isolation	S'exécute en tant que processus et a donc besoin d'un daemon qui gère les services des conteneurs, si ce daemon est tué tous les conteneurs le sont aussi	Les images sont toutes signées et vérifiées par un centre de sécurité centralisé et géré par Rocket
Taille et portabilité	Ne gère bien la portabilité que sur des machines Linux	Meilleure dans le domaine	Partage et découverte d'image simple et sans authentification
Simplicité à manager	Beaucoup d'outils de management déjà développé et possibilité de créer	Les outils de management externe sont bien développés, la	Outils de management avancés avec possibilité de mettre

	des conteneurs en tant qu'utilisateur	communauté juge que c'est la meilleure solution pour de Devops, il faut un accès administrateur (sudo)	plusieurs services dans un seul conteneur
Facilité d'utilisation	Facile à prendre en main et ligne de commande simple à utiliser	Facile à prendre en main et ligne de commande simple à utiliser, de plus les images avec les applications souhaitées sont souvent déjà disponibles grâce à la communauté	Un peu plus difficile à utiliser car la stratégie d'utilisation repose plus sur le "from scratch"

3. Similitudes et différences entre les architectures d'hyperviseurs de type 1 et de type 2

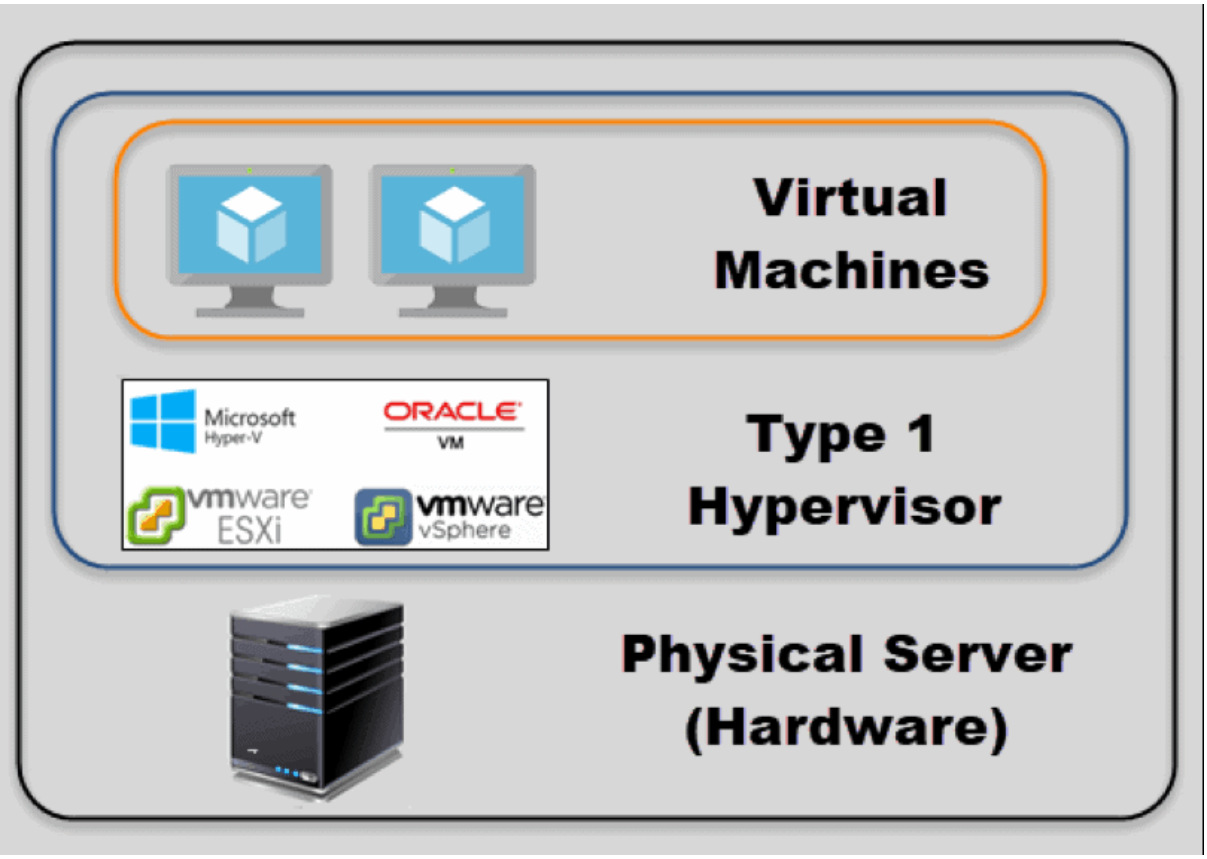
Il existe deux types d'hyperviseurs sur lesquels fonctionnent les machines virtuelles. Voici une comparaison des deux types :

	Type 1 hypervisor	Type 2 hypervisor
AKA	Bare-metal ou native	Hébergé au niveau de l'hôte
Définition	Fonctionne sur le système avec les VM fonctionnant directement sur eux (hyperviseur)	Fonctionne sur un OS classique
Virtualisation	Virtualisation matériel (hardware)	Virtualisation de l'OS
Scalabilité (*capacité d'un système à continuer de fonctionner de manière normale lorsque le nombre d'utilisateurs augmente*)	Meilleur type	Dépendant de l'OS dans lequel il tourne
Installation/Configuration	Simple tant qu'il y a le matériel nécessaire	Configuration bien plus simple étant donné qu'il y a déjà l'OS
Performance	Meilleure performance car il n'y a pas de couches intermédiaires	Dépendant de l'OS donc performance affectée
Vitesse	Plus rapide	Plus lent car dépendant de

		l'OS
Sécurité	Mieux sécurisé	Moins sécurisé car s'il y a un problème sur l'OS de base, tout le système sera impacté y compris l'Hyperviseur protégé.
Exemples	• VMware ESXi • Microsoft Hyper-V	• VirtualBox • VMWare Workstation player

4. Différence entre les deux principaux modes de connexion réseau pour les hôtes de virtualisation

Comme vu dans le tableau plus haut, **VirtualBox** est un hyperviseur de type 2.
Openstack ne rentre dans aucune des deux catégories par défaut car on va pouvoir choisir avec quel type d'hyperviseur on veut travailler. En effet, Openstack peut être utilisé pour créer des cloud privés comme publics, ce n'est pas un hyperviseur en soit, c'est un logiciel qui traduit en solution de cloud offrant ainsi la possibilité d'utiliser de nombreux hyperviseurs différents.

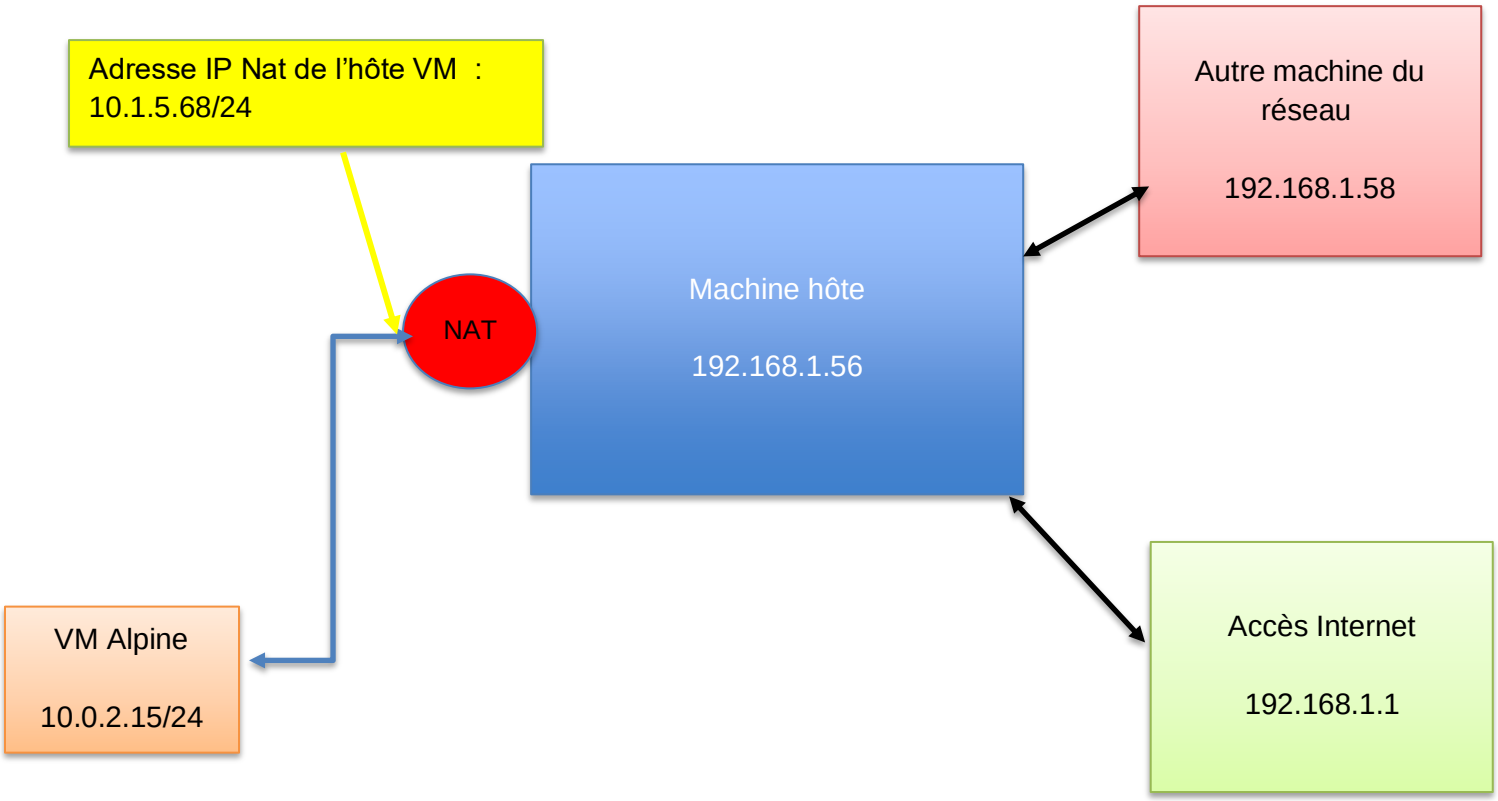


Objectifs 4 et 5

1. Création de la VM et configurations

Pour mener à bien cette partie, nous avons suivi pas à pas les différentes instructions inscrites sur le sujet sans rencontrer de problème particulier.

2. Test de la connectivité



Adresse IP dans la VM obtenu de l'hôte : 10.0.2.15/24 (par DHCP du logiciel de virtualisation)
Passerelle par défaut dans la VM obtenu par l'hôte : 10.1.5.68/24

En mode NAT, la VM va utiliser la translation d'adresse, la machine hôte servant de passerelle et effectuant la translation d'adresse.

La machine hôte effectue une translation d'adresse avant d'envoyer les paquets de la VM vers le réseau. Elle met son adresse IP en source du paquet et tient à jour une table de translation.

Une fois la réponse reçue, la machine hôte sait que le paquet est à destination de la VM et met celui-ci à jour en conséquence avant de le transmettre à la VM.
Dans la VM, un ping sur 10.1.5.68 répond, un ping sur 192.168.1.14 aussi car ces deux adresses correspondent aux 2 cartes réseau de l'hôte, l'une réelle (192.168.1.56) et l'autre virtuelle créer par VirtualBox (DHCP : 10.0.2.15).

Un ping de la VM vers 192.168.1.58 répond, l'hôte fait bien passerelle.

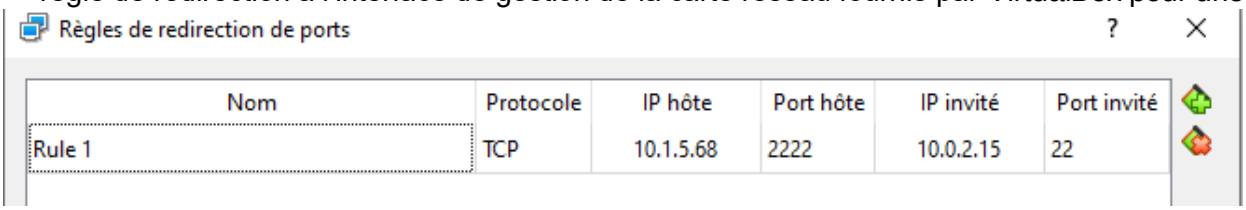
L'accès à Internet se fera de l'adresse 10.0.2.15 vers la passerelle 10.1.5.68, NAT entre l'adresse 10.1.5.68 et 192.168.1.56, puis passerelle du réseau 192.168.1.1.

Un autre poste que l'hôte ne pourra pas accéder à la VM. La carte réseau virtuelle de l'hôte ne comporte pas de passerelle vers les autres ordinateurs, uniquement une passerelle entre lui et ses VMs.

Si plusieurs machines doivent communiquer avec la VM, il faut utiliser le mode pont (ou modifier les réglages réseau, ce qui implique une certaine maîtrise et sort du cadre de la configuration par défaut). Un ping de l'hôte (192.168.1.58) vers l'adresse IP de la VM 10.0.2.15 ne fonctionne pas.

3. Redirection de port

Dans cette partie, nous allons mettre en place le « port Forwading » en ajoutant une nouvelle règle de redirection à l'interface de gestion de la carte réseau fournie par VirtualBox pour une



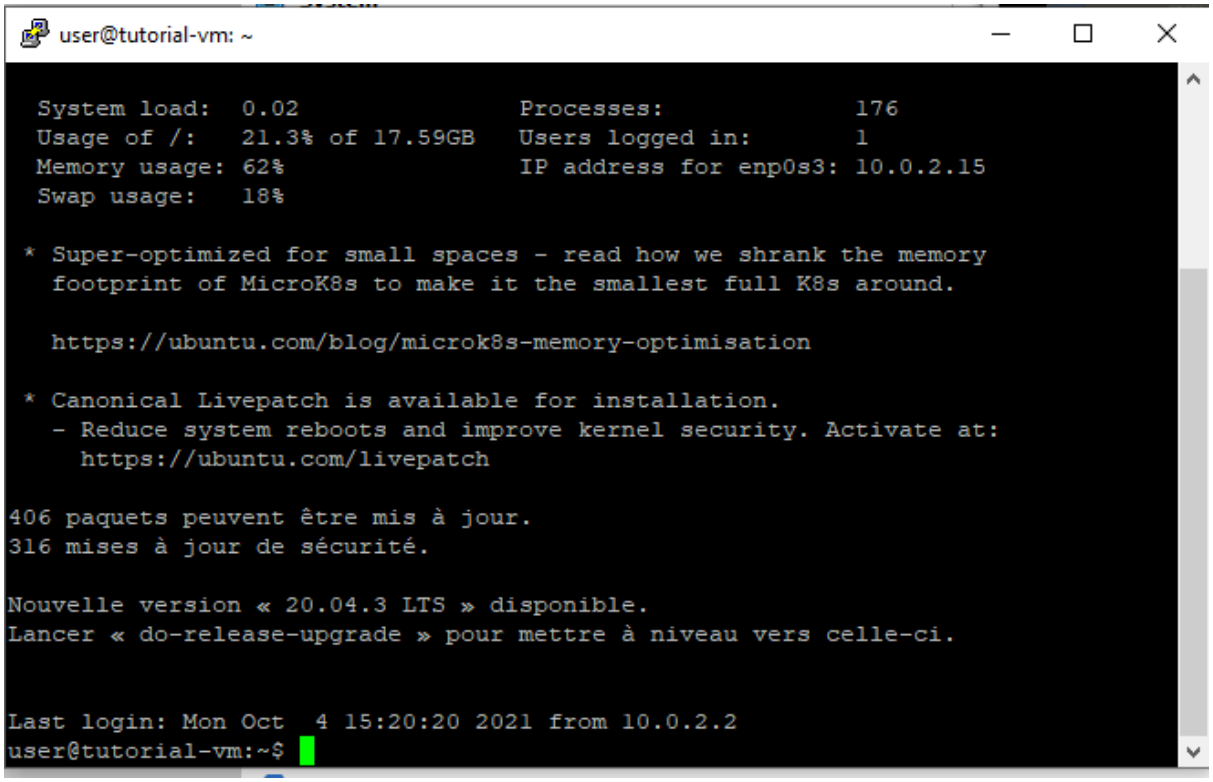
application dédiée en l'occurrence SSH.

Voici la règle que nous avons ajouté dans les réglages de VirtualBox associé à la VM :

Le SSH fonctionne maintenant bien de notre hôte à notre VM.

De plus, nous avons utilisé cette commande pour nous connecter à notre VM par SSH :

Après avoir rentré « user et mot de passe », on obtient alors :



En faisant correspondre le port de la VM à celui de l'hôte, toute personne souhaitant accéder au port 22 de notre VM doit simplement accéder au port 2222 de notre hôte (en envoyant le paquet à l'IP de l'hôte), et l'hôte traduira l'adresse IP.

4. Duplication de la VM

Cette partie ne nous a posé aucun problème. Nous sommes parvenus sans difficulté à cloner la VM.

5. Provisionnement des conteneurs Docker

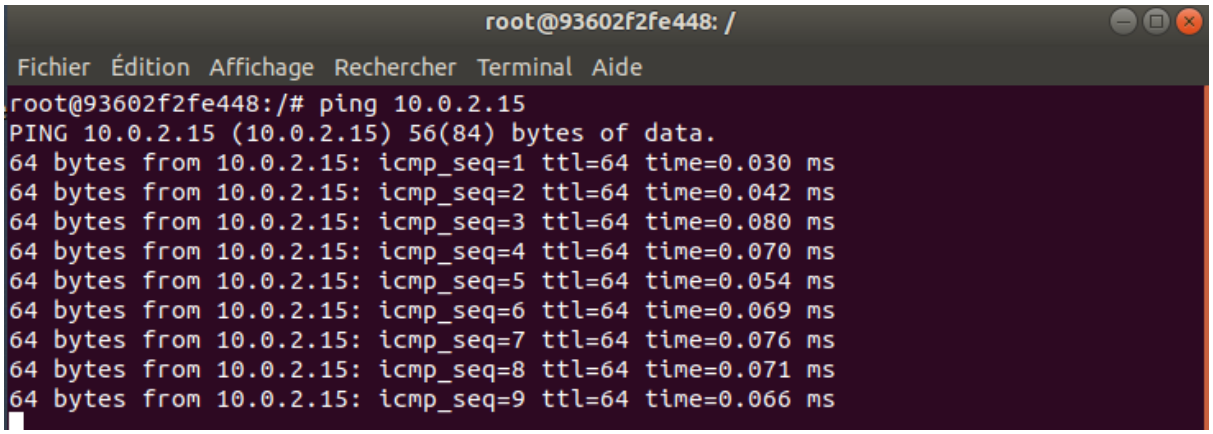
Par défaut, le conteneur se voit attribuer une adresse IP pour chaque réseau Docker auquel il se connecte. Et chaque réseau est créé avec un masque de sous-réseau par défaut, l'utilisant plus tard comme un pool pour donner les adresses IP.

Habituellement, Docker utilise le sous-réseau par défaut 172.17.0.0/16 pour la mise en réseau des conteneurs. Pour notre cas, l'adresse IP de notre conteneur « ct1 » est : **172.17.0.1**

Un ping du conteneur « ct1 » vers Internet (8.8.8.8) répond, l'hôte fait bien passerelle :

```
root@93602f2fe448:/# ping 8.8.8.8
PING 8.8.8.8 (8.8.8.8) 56(84) bytes of data.
64 bytes from 8.8.8.8: icmp_seq=1 ttl=113 time=6.85 ms
64 bytes from 8.8.8.8: icmp_seq=2 ttl=113 time=7.09 ms
64 bytes from 8.8.8.8: icmp_seq=3 ttl=113 time=7.11 ms
64 bytes from 8.8.8.8: icmp_seq=4 ttl=113 time=7.72 ms
64 bytes from 8.8.8.8: icmp_seq=5 ttl=113 time=7.64 ms
```

De même, un ping du conteneur vers la VM répond aussi :



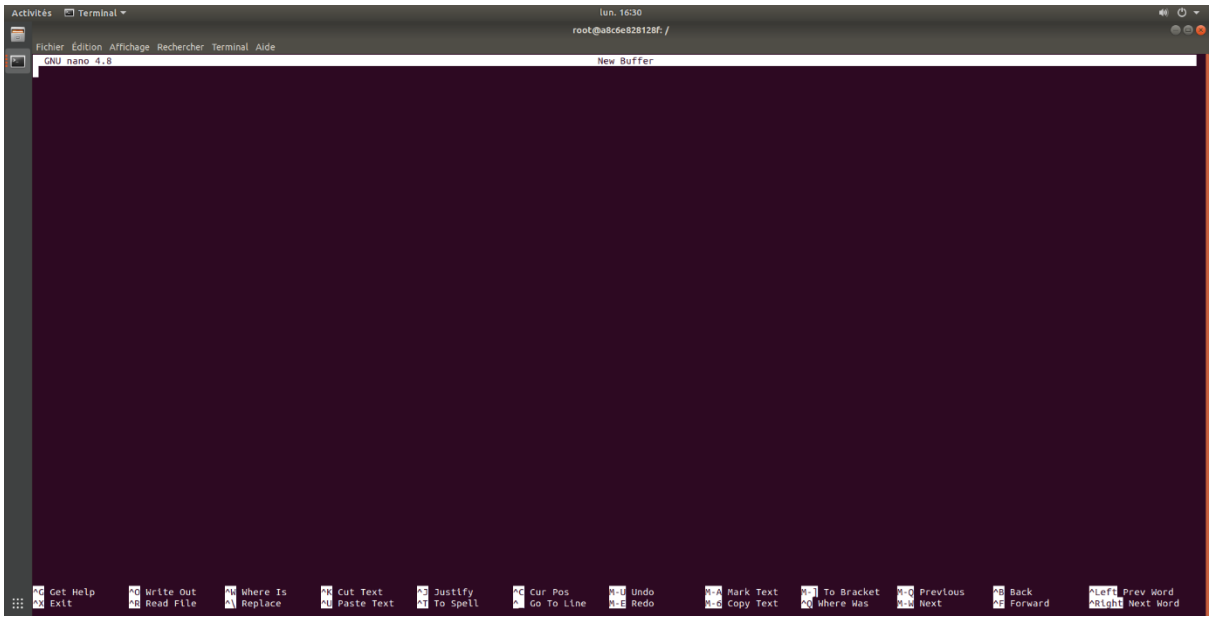
Enfin, un ping de la VM vers le conteneur « ct1 » marche aussi.

La machine hôte crée un réseau virtuel et une interface virtuelle dans ce réseau.

Ensuite, nous avons créé un autre conteneur « ct2 », basée sur l'image Ubuntu auquel on a mappé le port SSH (22) au port 2223 et installé l'éditeur de texte « nano ». Après cela, nous avons enregistré notre conteneur en créant une nouvelle image avec la commande « docker commit » :

```
root@tutorial-vm:/home/user# docker images
REPOSITORY          TAG                 IMAGE ID            CREATED             SIZE
ubuntu              version_finale      05db0da7844b        51 seconds ago      105MB
```

A partir de cela, nous avons créé un conteneur « ct3 » basé sur cette image auquel est installé nano :



Enfin, nous avons exploré une autre façon de créer un conteneur qui est le : Dockerfile.

En effet, il se présente de la façon suivante :

```
root@tutorial-vm:/home/user/Bureau/Docker# docker build -t ubuntu:latest .
Sending build context to Docker daemon  2.048kB
Step 1/4 : FROM ubuntu:latest
latest: Pulling from library/ubuntu
Digest: sha256:44ab2c3b26363823dcb965498ab06abf74a1e6af20a732902250743df0d4172d
Status: Downloaded newer image for ubuntu:latest
 ---> 597ce1600cf4
Step 2/4 : RUN apt update -y
 ---> Running in e0bcfb04f35d
```

Puis nous exécutons cette commande qui va lire et exécuter toutes les instructions dans le Dockerfile :

```
#-------Couche_OS ---------
FROM ubuntu:latest
RUN apt update -y
#---------------------------

#-----Installation_de_Nano--
RUN apt install -y nano
#---------------------------

#-----Shell-----------------
CMD ["bin/bash"]
#---------------------------
```

On obtient alors une image crée à partir de ce Dockerfile. Donc si nous créons un nouveau conteneur à partir de cette image, nous aurons nano installé dessus.

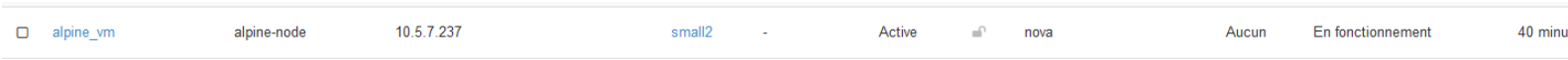
Objectifs 6 et 7

1. Création de machines virtuelles avec OpenStack

Nous avons créé des machines virtuelles en procédant de la façon suivante :

- Création d'un réseau privé *10.5.7.0/24* sur lequel on va connecter notre machine virtuelle ainsi que d'un routeur **RT1** liant le réseau privé et le réseau public

- Création de la machine Virtuelle basée sur une image Alpine



- Configuration des règles de sécurités pour autoriser les protocoles suivants : ICMP, SSH, TCP

Gérer les règles du groupe de sécurité : default (74fa2b72-89de-445e-98e4-81550f97207e)

+ Ajouter une règle | Supprimer les Ré

Affichage de 9 éléments

Direction	Ether Type	IP Protocol	Port Range	Remote IP Prefix	Remote Security Group	Description	Actions
Sortie	IPv4	Tous	Tous	0.0.0.0/0	-	-	Supprimer une Ré
Sortie	IPv4	ICMP	Tous	0.0.0.0/0	-	-	Supprimer une Ré
Sortie	IPv4	TCP	Tous	0.0.0.0/0	-	-	Supprimer une Ré
Sortie	IPv6	Tous	Tous	::/0	-	-	Supprimer une Ré
Entrée	IPv4	Tous	Tous	-	default	-	Supprimer une Ré
Entrée	IPv4	ICMP	Tous	0.0.0.0/0	-	-	Supprimer une Ré
Entrée	IPv4	TCP	Tous	0.0.0.0/0	-	-	Supprimer une Ré
Entrée	IPv4	TCP	22 (SSH)	0.0.0.0/0	-	-	Supprimer une Ré
Entrée	IPv6	Tous	Tous	-	default	-	Supprimer une Ré

2. Test de connectivité

Selon la topologie de réseau, on a donc 2 réseaux : un réseau public (192.168.37.0/24) et un réseau privé (10.5.7.0/24). Les adresses IP attribuées sont les suivantes :

- Adresse IP de la machine virtuelle : 10.1.5.85
- Adresse IP de l'hôte (PC local) : 192.168.37…

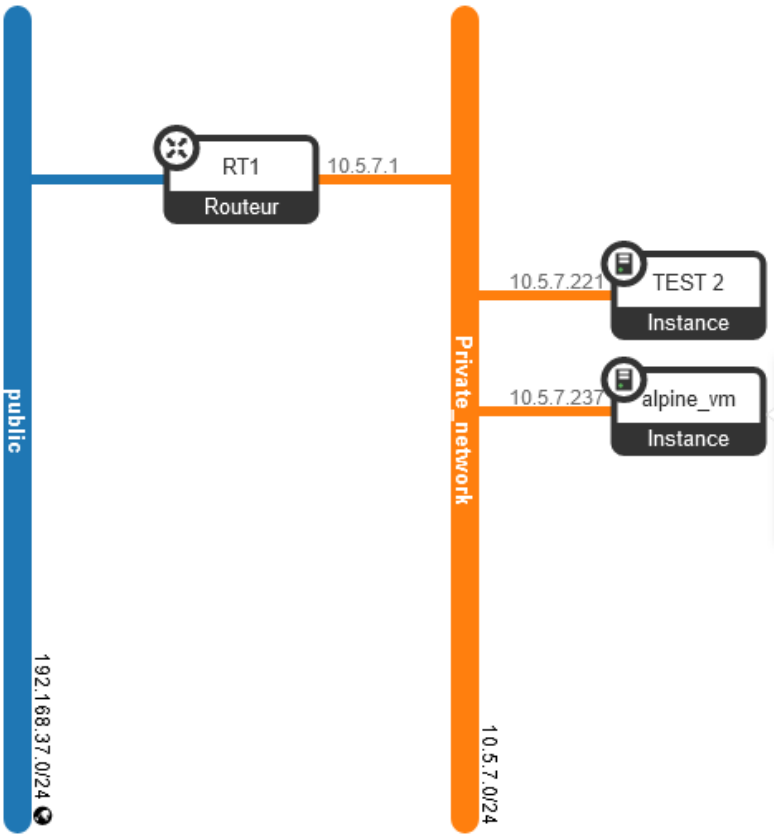
La connectivité fonctionne de la VM vers l'hôte ou de l'hôte vers la VM ne fonctionne pas car 192.168.37…. est sur un réseau privé d'OpenStack.

Pour rendre notre VM accessible, nous devons d'abord mettre en place un routeur passerelle connecté à la fois au réseau public et au réseau privé. Cela permet la connectivité de la VM vers le public (principe du NAT).

Nous avons ensuite demandé une adresse flottante sur le réseau public et l'avons liée à notre VM. Cela a rendu possible la connectivité du public vers la VM (via son adresse flottante).
Voici notre VM « TEST 2 », rattaché au réseau privé 10.5.7.221, auquel on a associé l'adresse IP flottante 192.168.37.44 :

☐	TEST 2	alpine-node	10.5.7.221, 192.168.37.44	small2	-

En outre, voici la topologie des réseaux mis en place :



3. Snapshot, restauration et redimensionnement d'une VM

Nous avons redimensionné la VM sans éteindre la machine. Cela a fonctionné mais nous avons été déconnectés de la console. Nous avons redimensionné la VM après l'extinction et cela a fonctionné aussi.

Cela montre la flexibilité d'OpenStack (le redimensionnement avec la machine éteinte ou allumée) en opposition à la VirtualBox (on doit arrêter la VM).

Cependant il y a une limitation : on ne peut pas redimensionner la VM. Pour résoudre ce problème nous pouvons cloner la VM et utiliser cette nouvelle VM pendant que nous éteignons et redimensionnons notre VM originale.

Objectifs 8 et 9

1. Installation du client OpenStack

Fait, nous n'avons eu aucun problème avec cette tâche.

2. Topologie et spécification des applications Web

Nous avons testé le service sur une VM Ubuntu locale et il fonctionne comme prévu.

3. Déployer l'application Calculator sur OpenStack

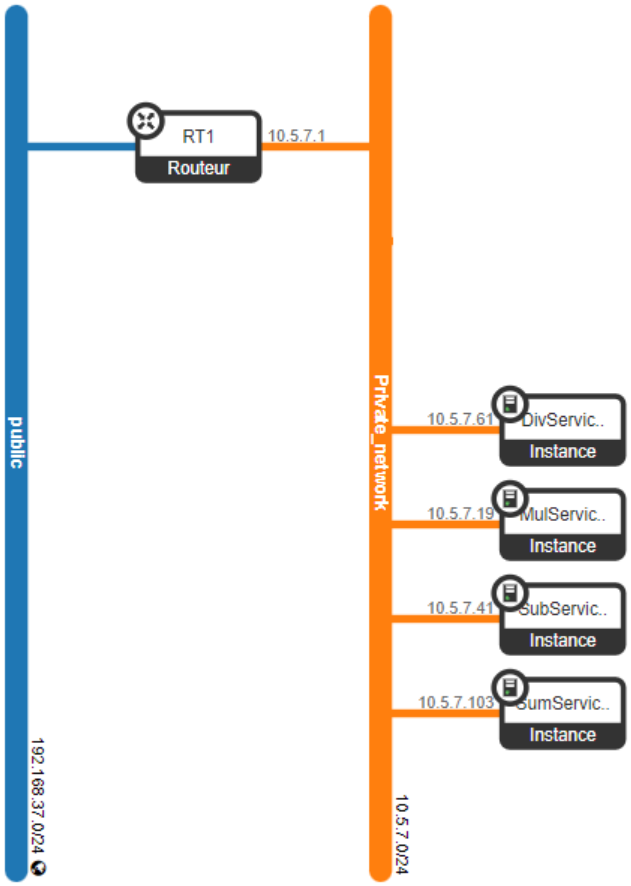
Nous avons créé 5 VMs : SumService, SubService, MulService, DivService et CalculatorService (CS).

Nous avons placé toutes nos machines dans le même réseau (10.5.7.0/24) et nous nous sommes assurés que la VM contenant le service de calculateur (CS) avait l'adresse IP de tous les autres sous-services.

Chaque service écoute sur un port qui se situe entre 50000 et 50050. En l'occurrence, on a la configuration suivante :

- SumService → port 50001
- SubService → port 50002
- MulService → port 50003
- DivService → port 500004
- CalculatorService → port 50000

On a donc la topologie suivante :



Nous avons donc testé nos services, en faisant un Curl à notre CS.

```
user@tutorial-vm:~$ curl -d "(5+6)*2" -X POST http://10.5.7.253:50000
result = 22
```

Ce qui donne dans l'affichage suivant :

```
user@tutorial-vm:~/Bureau$ node CalculatorService.js
Listening on port : 50000
New request :
(5+6)*2 = 22
```

4. Automatiser/Orchestrer le déploiement de l'application

Nous avons ensuite fait la même chose avec Docker et avons automatisé le déploiement de l'application.

Dans un premier temps, nous créons 5 fichiers pour *build* l'image custom docker dont nous avons besoin pour run nos micro-service. Voici le code du fichier Dockerfile que nous avons rédigé pour la création du calculateur :

```
FROM node:12.18.1
WORKDIR /app
USER root
RUN npm install sync-request
COPY ./CalculatorService.js .
CMD ["node", "CalculatorService.js"]
```

Voici une description des instructions utilisées dans ce fichier :

- *FROM node :12.18.1* → définit l'image de base qui inclut nodejs et npm
- *WORKDIR /app* → spécifie le répertoire de travail
- *USER root* → désigne quel est l'utilisateur (root) qui lancera les prochaines instructions RUN, CMD
- *RUN npm install sync-request* → permet d'installer le package « sync-request » indispensable au bon fonctionnement du calculateur.
- *COPY ./CalculatorService.js .* → Copie le fichier JavaScript dans le working directory.
- *CMD [''node'', ''CalculatorService.js'']* → enfin, nous spécifions la commande à lancer au démarrage du container ''node CalculatorService.js''.

Par analogie, les 4 autres dockerfiles sont rédigés de la même façon à l'exception de l'installation du package "sync-request" :

```
FROM node:12.18.1
WORKDIR /app
USER root
RUN npm install sync-request
COPY ./ DivService.js .
CMD ["node", "DivService.js"]
```

Nous avons créé un script *bash* pour construire toutes les images (quatre sous-services) et lancer les sous-services.

Pour s'assurer que les sous-services sont accessibles depuis le service de calculatrice, nous mappons leur port (port sur lequel le service se lance par défaut) au port de l'hôte.

```
#!/bin/bash

echo "Construction des images "

cd add
docker build -t add:12.18.1 -f Sum.dockerfile .

cd ../sub
docker build -t sub:12.18.1 -f Sub.dockerfile .

cd ../div
docker build -t div:12.18.1 -f Div.dockerfile .

cd ../mul
docker build -t mul:12.18.1 -f Mul.dockerfile .


echo "Run containers"

docker run -p 50001:50001 add:12.18.1 &
docker run -p 50002:50002 sub:12.18.1 &
docker run -p 50003:50003 mul:12.18.1 &
docker run -p 50004:50004 div:12.18.1 &
```

Avant de lancer le service calculateur, nous modifions le service Javascript pour nous assurer qu'il sache où trouver les sous-services (sur l'adresse de l'hôte, sur le port mappé). Afin de connaître leurs adresses IP, nous listons les containers docker avec la commande *"sudo docker container ps"* afin de récupérer l'ID des containers.

Puis à l'aide de la commande *"sudo docker inspect [id container]"* nous récupérons l'adresse IP de chaque conteneur :

```
var http = require ('http');
var request = require('sync-request');

const PORT = process.env.PORT || 50005;

const SUM_SERVICE_IP_PORT = 'http://172.17.0.3:50001';
const SUB_SERVICE_IP_PORT = 'http://172.17.0.4:50002';
const MUL_SERVICE_IP_PORT = 'http://172.17.0.5:50003';
const DIV_SERVICE_IP_PORT = 'http://172.17.0.2:50004';
```

Nous lançons ensuite le service Calculatrice et mappons également son port pour nous assurer qu'il est accessible depuis notre PC hôte :

```
root@yacine-VirtualBox:/home/yacine/Bureau/Cloud/calcul# docker run -p 50005:50005 cal:12.18.1
Listening on port : 50005
```

Voici la liste de tous les conteneurs lancés :

```
root@yacine-VirtualBox:/home/yacine/Bureau/Cloud/calcul# docker ps
CONTAINER ID   IMAGE         COMMAND                  CREATED          STATUS          PORTS
8d5b801c8bb3   cal:12.18.1   "docker-entrypoint.s…"   6 minutes ago    Up 6 minutes    0.0.0.0:50005->50005/tcp, :::50005->50005/tcp
613b95a70dde   mul:12.18.1   "docker-entrypoint.s…"   24 minutes ago   Up 24 minutes   0.0.0.0:50003->50003/tcp, :::50003->50003/tcp
e78f95dbe65a   add:12.18.1   "docker-entrypoint.s…"   24 minutes ago   Up 24 minutes   0.0.0.0:50001->50001/tcp, :::50001->50001/tcp
995eeb9f3316   sub:12.18.1   "docker-entrypoint.s…"   24 minutes ago   Up 24 minutes   0.0.0.0:50002->50002/tcp, :::50002->50002/tcp
d38eb2ddc20e   div:12.18.1   "docker-entrypoint.s…"   24 minutes ago   Up 24 minutes   0.0.0.0:50004->50004/tcp, :::50004->50004/tcp
```

Ainsi, nous parvenons finalement à réaliser des opérations appelant l'ensemble des micro-services et à retourner le résultat :

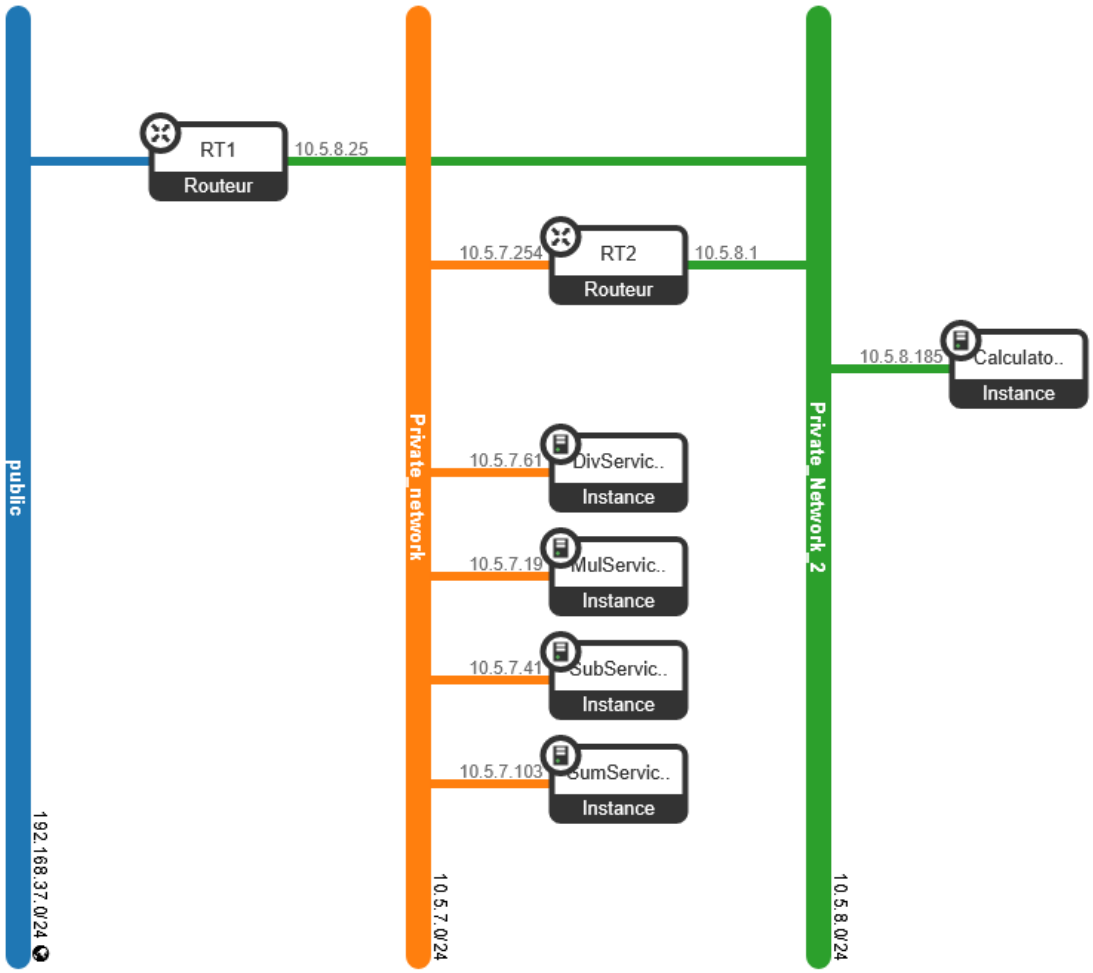
```
root@yacine-VirtualBox:/home/yacine/Bureau/Cloud/calcul# curl -d "((5+6)+(2*5)+(10/2))" -X PO
ST http://172.17.0.6:50005
result = 26
```

Objectifs 10 et 11

Le dernier objectif est de concevoir et d'implémenter la réponse à fournir à un client demandeur d'une topologie de nœuds incluant des fonctions de communication de niveau 3 (réseau) à 7 (application).

Nous plaçons nos sous-services sur le réseau 10.5.7.0/24, et notre service Calculatrice principale sur son propre réseau 10.5.8.0/24 .

Ces deux réseaux sont inter-accessibles par le biais d'un routeur directement connecté aux deux réseaux (RT2).



Afin que les sous-réseaux puissent communiquer entre eux, nous créons une route entre @ du réseau 10.5.7.0/24 et le routeur RT2. De même, nous créons une route entre @ du réseau 10.5.8.0/24 et le routeur RT1 sur chacun sous-services.

Voici la commande `route add -net 10.5.7.0/24 gw 10.5.8.1` sur RT1 et `route add -net 10.5.8.0/24 gw 10.5.7.254` sur chacun des sous-services.

Pour rendre le service *CalculatorService* accessible depuis l'extérieur, nous avons associé une adresse IP flottante (accessible au public) à notre VM.

Instance Name	Image Name	IP Address
Calculator_Service_New	Calculator_Service_New	10.5.8.185, 192.168.37.106
DivService	ubuntu4CLV	10.5.7.61
MulService	ubuntu4CLV	10.5.7.19
SubService	ubuntu4CLV	10.5.7.41
SumService	ubuntu4CLV	10.5.7.103

Après avoir effectué une connexion SSH à l'aide de PuTTY, nous avons testé le CalculatorService et on a obtenu le résultat suivant :

```
root@tutorial-vm:/home/user# node CalculatorService.js
Listening on port : 50000
New request :
5+6 = 11
```