tiny.cc/TP_Cloud

1. Similarities and differences between the main virtualisation hosts (VM et CT)

- Differences between VM and Container : Each VM has its own guest OS and its own Bins/Libs environment while each container has its own Bins/Libs environment but not its own Guest OS.
- From an administrator point of view :
    - VMs are more costly in ressources because there are more layers of abstraction layers between apps and hardware
    - Administrators can modify each VM access rights to resources more easily because each VM has its guest OS.
    - In a VM environment, the security of the app is greater because the app is the only one running on its guest OS while in a container environment, the apps are running in the same OS. From an admin point of view, the security of his ressources is also more guaranteed with VMs because apps are not running on the server's OS.

- From a developper point of view :
    - In a VM environment, the developer knows that the memory and CPU resources can be adapted to his needs while in a container he has to take into consideration the limited resources that are available.
    - In a VM environment, the security of the app is greater because the app is the only one running on its guest OS while in a container environment, the apps are running in the same OS.
    - In a VM environment, response time is greater and less predictable because there are more layers of abstraction layers between the app and the hardware.
    - Containers are more useful for CI because containers are natively made for CI and devops tools.
    - developer wants to change easily environment to test several use cases

In conclusion, developers like CT while admins like VM

2. Similarities and differences between the existing CT types

LXC has built-in controles groups (cgroups), each cgroup offers apps a complete ressources isolation without the need of VM. It is an advantage from a multi-tenancy point of view since user IDs also are completely isolated

- Application isolation and resources, from a multi-tenancy point of view:
  Docker is better from a multi-tenancy point of view because runtime issues are dealt with separately for each app, whereas on lxc applications are not isolated from one another.
- Containerization level (e.g. operating system, application) :
  The containerization is closer to the app in Docker
- Tooling (e.g. API, continuous integration, or service composition) :
  A docker can only run multiple instances of the same app while an LXC can

run several apps simultaneously

-A docker has a “Docker Engine” which has its own kernel running its own OS. This means a docker running Ubuntu could work on a different host OS, Windows for instance.

Docker can be seen as an extension of Lxc’s capabilities. It is specialized application virtualization individually.

docker is known for having some security issues, RKT is a more secure container technologie

3. Similarities and differences between Type 1 & Type 2 of hypervisors’ architectures

Type-1 hypervisors run directly on the host's hardware to control the hardware and to manage guest operating systems.

Types-2 hypervisors run on a conventional operating system (OS) just as other computer programs do. Types-2 hypervisors can even run next to another process. Virtual Box is a Type-2 hypervisor.

An Open stack is an example of Type-1 hypervisor.

Practical part

(04/10/21) session 2
<u>Sur la VM :</u>

```
user@tutorial-vm:~$ ifconfig
enp0s3: flags=4163<UP,BROADCAST,RUNNING,MULTICAST>  mtu 1500
        inet 10.0.2.15  netmask 255.255.255.0  broadcast 10.0.2.255
        inet6 fe80::a00:27ff:fe6b:f65b  prefixlen 64  scopeid 0x20<link>
        ether 08:00:27:6b:f6:5b  txqueuelen 1000  (Ethernet)
        RX packets 10393  bytes 15574906 (15.5 MB)
        RX errors 0  dropped 0  overruns 0  frame 0
        TX packets 2389  bytes 159931 (159.9 KB)
        TX errors 0  dropped 0 overruns 0  carrier 0  collisions 0

lo: flags=73<UP,LOOPBACK,RUNNING>  mtu 65536
        inet 127.0.0.1  netmask 255.0.0.0
        inet6 ::1  prefixlen 128  scopeid 0x10<host>
        loop  txqueuelen 1000  (Boucle locale)
        RX packets 40  bytes 3758 (3.7 KB)
        RX errors 0  dropped 0  overruns 0  frame 0
        TX packets 40  bytes 3758 (3.7 KB)
        TX errors 0  dropped 0 overruns 0  carrier 0  collisions 0
```

Sur l'hôte :

```
U:\>ipconfig

Configuration IP de Windows


Carte Ethernet VirtualBox Host-Only Network :

   Suffixe DNS propre à la connexion. . . :
   Adresse IPv6 de liaison locale. . . . .: fe80::aded:bd28:4294:c2a3%6
   Adresse IPv4. . . . . . . . . . . . . .: 192.168.56.1
   Masque de sous-réseau. . . . . . . . . : 255.255.255.0
   Passerelle par défaut. . . . . . . . . :

Carte Ethernet Ethernet :

   Suffixe DNS propre à la connexion. . . : insa-toulouse.fr
   Adresse IPv6 de liaison locale. . . . .: fe80::99f0:4ea5:2e43:b4e5%5
   Adresse IPv4. . . . . . . . . . . . . .: 10.1.5.74
   Masque de sous-réseau. . . . . . . . . : 255.255.0.0
   Passerelle par défaut. . . . . . . . . : 10.1.0.254
```

We were able to ping the host machine from the VM by using the host's private address.

- Check the connectivity from the VM to the outside. What do you observe

We observe that the hosted VM sees its host and the rest of the local network

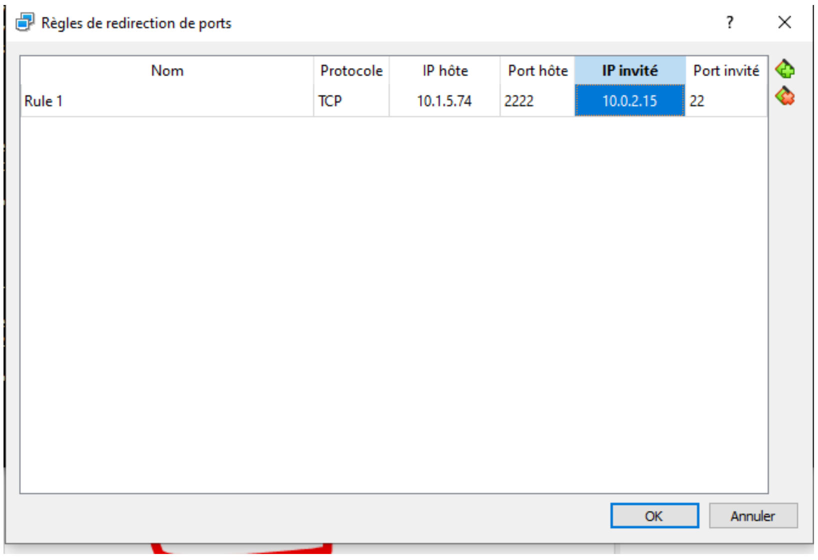
- Check the connectivity from your neighbour's host to your hosted VM.

- Check the connectivity from your host to the hosted VM.

As expected, the host cannot see the VM it's hosting

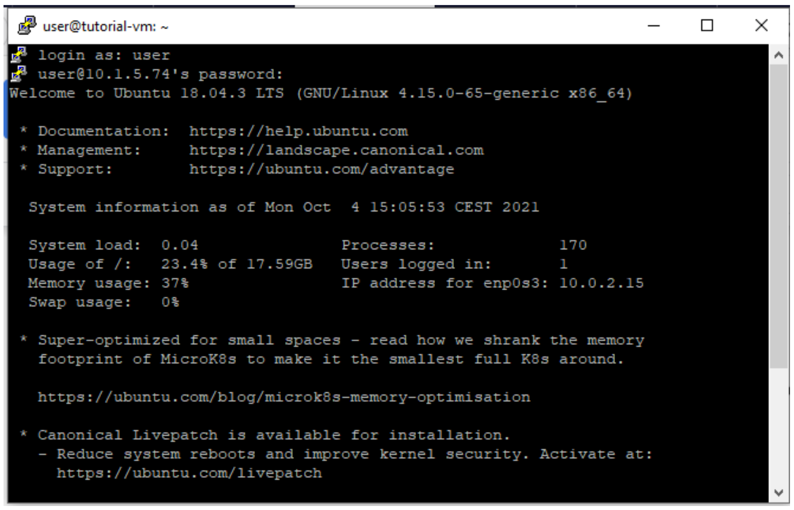
Third part: Set up the "missing" connectivity

At this point the host cannot reach the hosted VM(s).
To do that we have to create the path the host will take to reach the VM(s), we do that by using the SSH port of the VM.
we created the rule routing rule from the VM so that the ssh protocol reaches the port 22 of the VM.



Nom	Protocole	IP hôte	Port hôte	IP invité	Port invité
Rule 1	TCP	10.1.5.74	2222	10.0.2.15	22

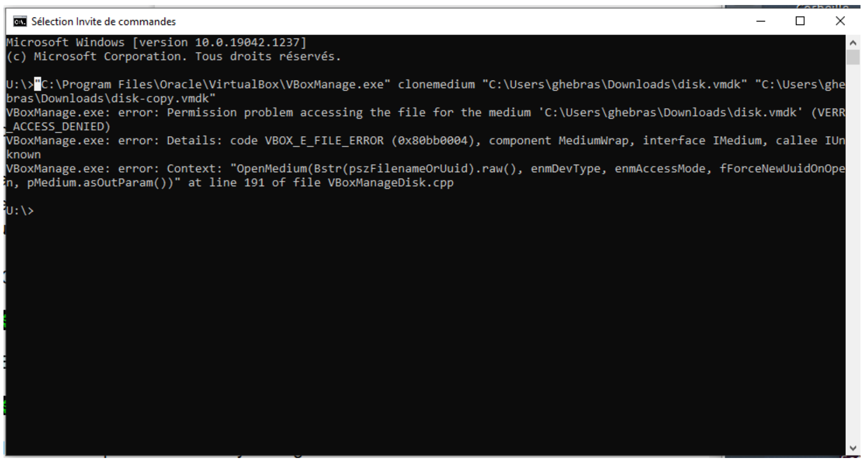
We then used PuTTy as a SSH client using the port 2222 and the host IP (10.1.5.74).



```
login as: user
user@10.1.5.74's password:
Welcome to Ubuntu 18.04.3 LTS (GNU/Linux 4.15.0-65-generic x86_64)

 * Documentation:  https://help.ubuntu.com
 * Management:     https://landscape.canonical.com
 * Support:        https://ubuntu.com/advantage

  System information as of Mon Oct  4 15:05:53 CEST 2021

  System load:  0.04               Processes:             170
  Usage of /:   23.4% of 17.59GB   Users logged in:       1
  Memory usage: 37%                IP address for enp0s3: 10.0.2.15
  Swap usage:   0%

 * Super-optimized for small spaces - read how we shrank the memory
   footprint of MicroK8s to make it the smallest full K8s around.

   https://ubuntu.com/blog/microk8s-memory-optimisation

 * Canonical Livepatch is available for installation.
   - Reduce system reboots and improve kernel security. Activate at:
     https://ubuntu.com/livepatch
```

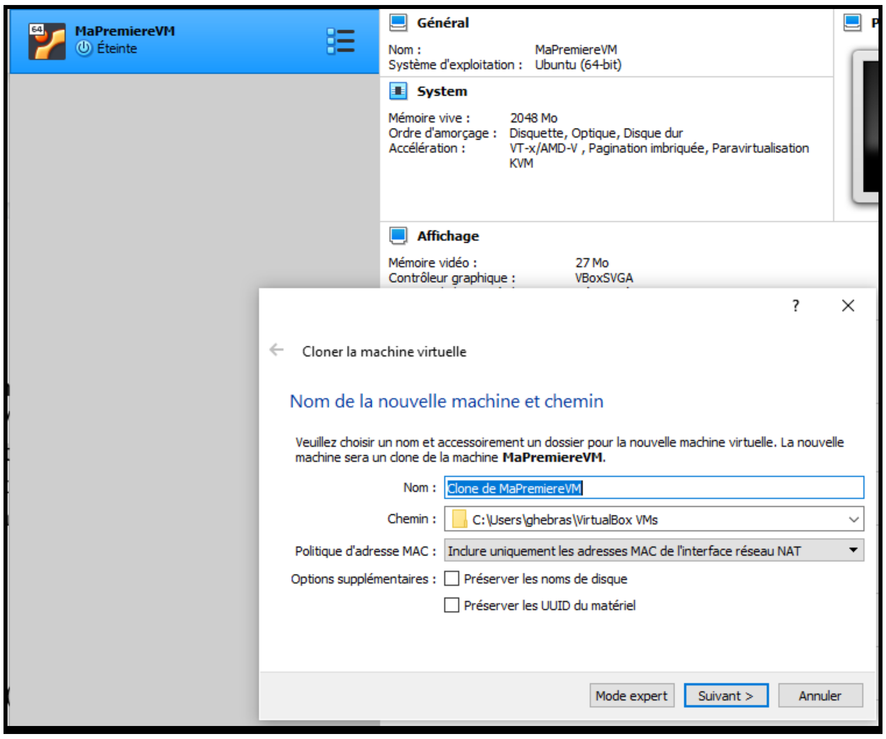
Fourth part: VM duplication

we tries using the following command:



```
Microsoft Windows [version 10.0.19042.1237]
(c) Microsoft Corporation. Tous droits réservés.

U:\>"C:\Program Files\Oracle\VirtualBox\VBoxManage.exe" clonemedium "C:\Users\ghebras\Downloads\disk.vmdk" "C:\Users\ghe
bras\Downloads\disk-copy.vmdk"
VBoxManage.exe: error: Permission problem accessing the file for the medium 'C:\Users\ghebras\Downloads\disk.vmdk' (VERR
_ACCESS_DENIED)
VBoxManage.exe: error: Details: code VBOX_E_FILE_ERROR (0x80bb0004), component MediumWrap, interface IMedium, callee IUn
known
VBoxManage.exe: error: Context: "OpenMedium(Bstr(pszFilenameOrUuid).raw(), enmDevType, enmAccessMode, fForceNewUuidOnOpe
n, pMedium.asOutParam())" at line 191 of file VBoxManageDisk.cpp

U:\>
```

but for an unknown reason this didn’t work. We had to simply use the included cloning function in VirtualBox:


MaPremiereVM
Éteinte

Général
Nom : MaPremiereVM
Système d'exploitation : Ubuntu (64-bit)

System
Mémoire vive : 2048 Mo
Ordre d'amorçage : Disquette, Optique, Disque dur
Accélération : VT-x/AMD-V , Pagination imbriquée, Paravirtualisation KVM

Affichage
Mémoire vidéo : 27 Mo
Contrôleur graphique : VBoxSVGA

Cloner la machine virtuelle

Nom de la nouvelle machine et chemin

Veuillez choisir un nom et accessoirement un dossier pour la nouvelle machine virtuelle. La nouvelle machine sera un clone de la machine **MaPremiereVM**.

Nom : Clone de MaPremiereVM
Chemin : C:\Users\ghebras\VirtualBox VMs
Politique d'adresse MAC : Inclure uniquement les adresses MAC de l'interface réseau NAT
Options supplémentaires : ☐ Préserver les noms de disque
☐ Préserver les UUID du matériel

Fifth part: Docker containers provisioning

ifconfig in docker :

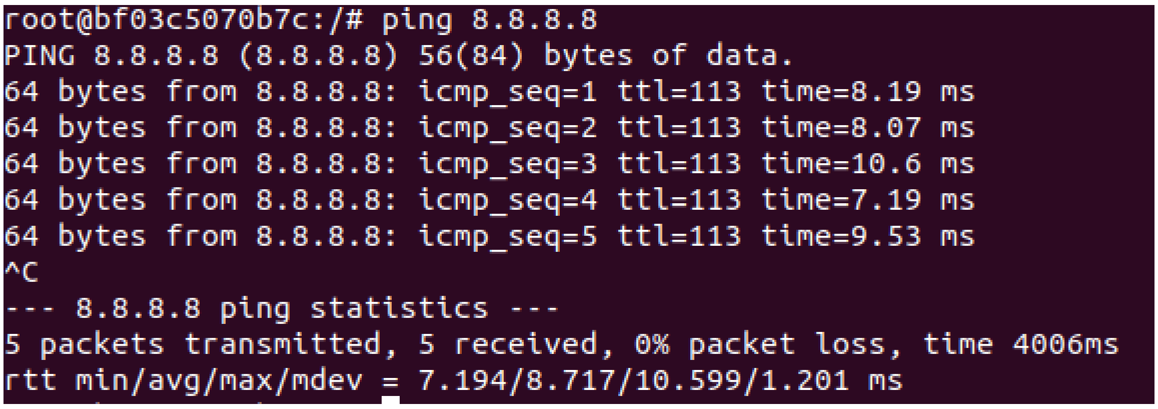
```
root@bf03c5070b7c:/# ifconfig
eth0: flags=4163<UP,BROADCAST,RUNNING,MULTICAST>  mtu 1500
        inet 172.17.0.2  netmask 255.255.0.0  broadcast 172.17.255.255
        ether 02:42:ac:11:00:02  txqueuelen 0  (Ethernet)
        RX packets 6687  bytes 19950348 (19.9 MB)
        RX errors 0  dropped 0  overruns 0  frame 0
        TX packets 4888  bytes 268993 (268.9 KB)
        TX errors 0  dropped 0 overruns 0  carrier 0  collisions 0

lo: flags=73<UP,LOOPBACK,RUNNING>  mtu 65536
        inet 127.0.0.1  netmask 255.0.0.0
        loop  txqueuelen 1000  (Local Loopback)
        RX packets 0  bytes 0 (0.0 B)
        RX errors 0  dropped 0  overruns 0  frame 0
        TX packets 0  bytes 0 (0.0 B)
        TX errors 0  dropped 0 overruns 0  carrier 0  collisions 0

root@bf03c5070b7c:/#
```

1. What is the Docker IP address.

172.17.0.2

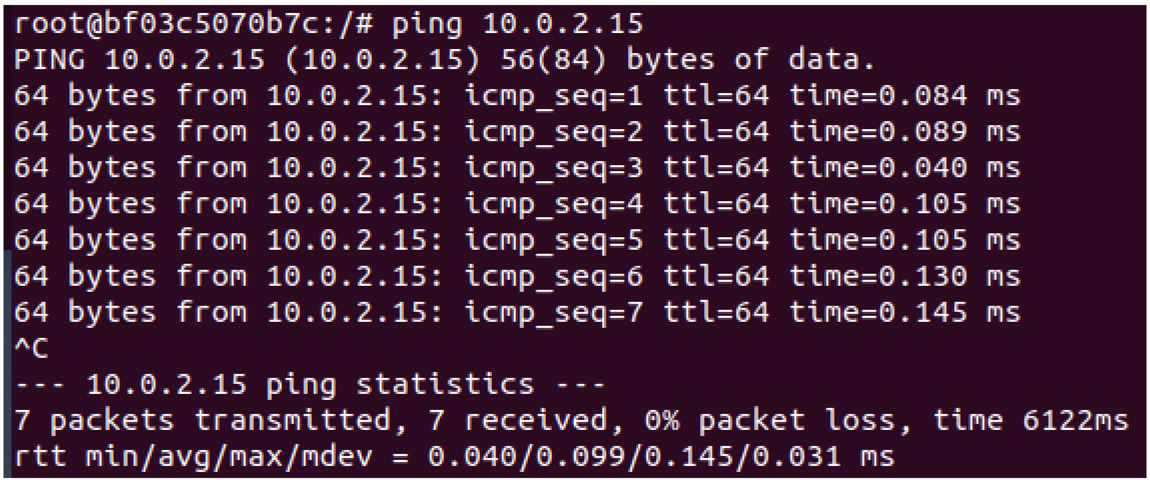
2. Ping an Internet resource from Docker

ping google from docker :



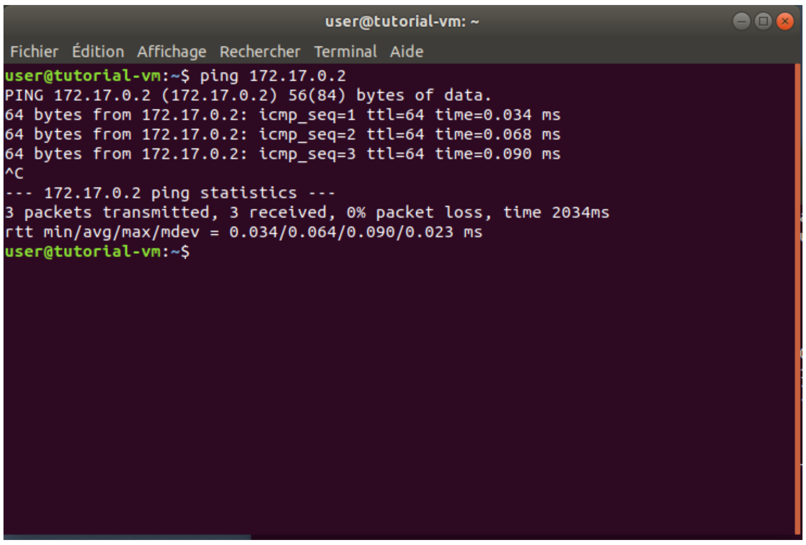
```
root@bf03c5070b7c:/# ping 8.8.8.8
PING 8.8.8.8 (8.8.8.8) 56(84) bytes of data.
64 bytes from 8.8.8.8: icmp_seq=1 ttl=113 time=8.19 ms
64 bytes from 8.8.8.8: icmp_seq=2 ttl=113 time=8.07 ms
64 bytes from 8.8.8.8: icmp_seq=3 ttl=113 time=10.6 ms
64 bytes from 8.8.8.8: icmp_seq=4 ttl=113 time=7.19 ms
64 bytes from 8.8.8.8: icmp_seq=5 ttl=113 time=9.53 ms
^C
--- 8.8.8.8 ping statistics ---
5 packets transmitted, 5 received, 0% packet loss, time 4006ms
rtt min/avg/max/mdev = 7.194/8.717/10.599/1.201 ms
```

3. Ping the VM from Docker



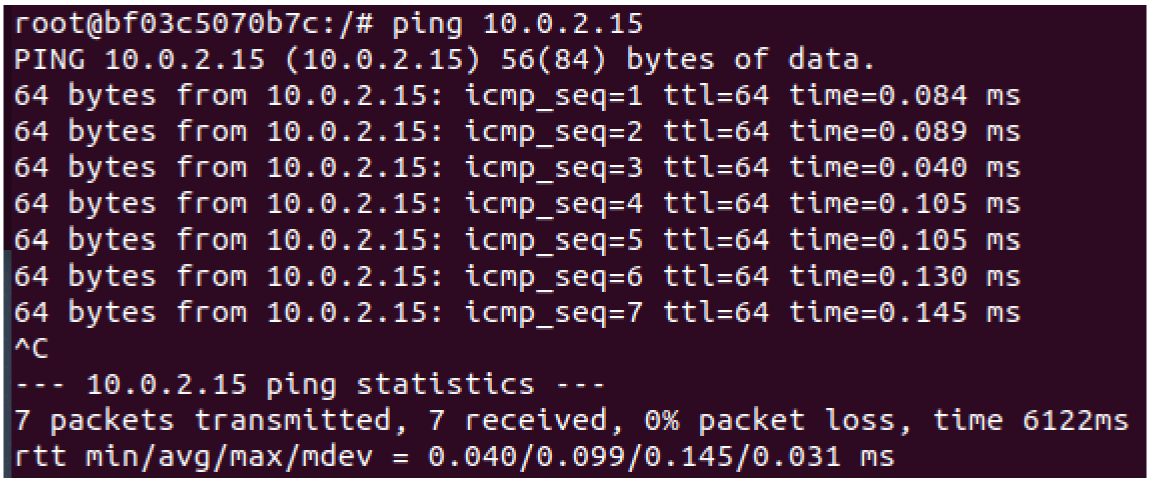
```
root@bf03c5070b7c:/# ping 10.0.2.15
PING 10.0.2.15 (10.0.2.15) 56(84) bytes of data.
64 bytes from 10.0.2.15: icmp_seq=1 ttl=64 time=0.084 ms
64 bytes from 10.0.2.15: icmp_seq=2 ttl=64 time=0.089 ms
64 bytes from 10.0.2.15: icmp_seq=3 ttl=64 time=0.040 ms
64 bytes from 10.0.2.15: icmp_seq=4 ttl=64 time=0.105 ms
64 bytes from 10.0.2.15: icmp_seq=5 ttl=64 time=0.105 ms
64 bytes from 10.0.2.15: icmp_seq=6 ttl=64 time=0.130 ms
64 bytes from 10.0.2.15: icmp_seq=7 ttl=64 time=0.145 ms
^C
--- 10.0.2.15 ping statistics ---
7 packets transmitted, 7 received, 0% packet loss, time 6122ms
rtt min/avg/max/mdev = 0.040/0.099/0.145/0.031 ms
```

3. Ping the Docker from the VM



```
user@tutorial-vm:~$ ping 172.17.0.2
PING 172.17.0.2 (172.17.0.2) 56(84) bytes of data.
64 bytes from 172.17.0.2: icmp_seq=1 ttl=64 time=0.034 ms
64 bytes from 172.17.0.2: icmp_seq=2 ttl=64 time=0.068 ms
64 bytes from 172.17.0.2: icmp_seq=3 ttl=64 time=0.090 ms
^C
--- 172.17.0.2 ping statistics ---
3 packets transmitted, 3 received, 0% packet loss, time 2034ms
rtt min/avg/max/mdev = 0.034/0.064/0.090/0.023 ms
user@tutorial-vm:~$
```

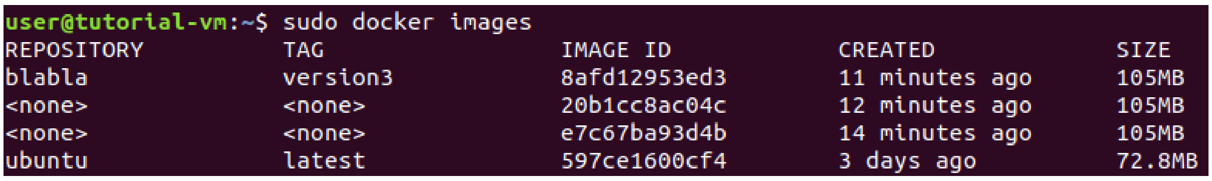
pinging vm from docker :



```
root@bf03c5070b7c:/# ping 10.0.2.15
PING 10.0.2.15 (10.0.2.15) 56(84) bytes of data.
64 bytes from 10.0.2.15: icmp_seq=1 ttl=64 time=0.084 ms
64 bytes from 10.0.2.15: icmp_seq=2 ttl=64 time=0.089 ms
64 bytes from 10.0.2.15: icmp_seq=3 ttl=64 time=0.040 ms
64 bytes from 10.0.2.15: icmp_seq=4 ttl=64 time=0.105 ms
64 bytes from 10.0.2.15: icmp_seq=5 ttl=64 time=0.105 ms
64 bytes from 10.0.2.15: icmp_seq=6 ttl=64 time=0.130 ms
64 bytes from 10.0.2.15: icmp_seq=7 ttl=64 time=0.145 ms
^C
--- 10.0.2.15 ping statistics ---
7 packets transmitted, 7 received, 0% packet loss, time 6122ms
rtt min/avg/max/mdev = 0.040/0.099/0.145/0.031 ms
```

To elaborate on those results, since docker is a CT we don’t have to do the whole routing rule in order to enable the two way link.

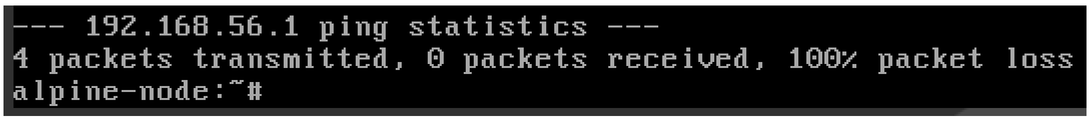
5 through 7:
the commit named blabla:



```
user@tutorial-vm:~$ sudo docker images
REPOSITORY          TAG                 IMAGE ID            CREATED             SIZE
blabla              version3            8afd12953ed3        11 minutes ago      105MB
<none>              <none>              20b1cc8ac04c        12 minutes ago      105MB
<none>              <none>              e7c67ba93d4b        14 minutes ago      105MB
ubuntu              latest              597ce1600cf4        3 days ago          72.8MB
```

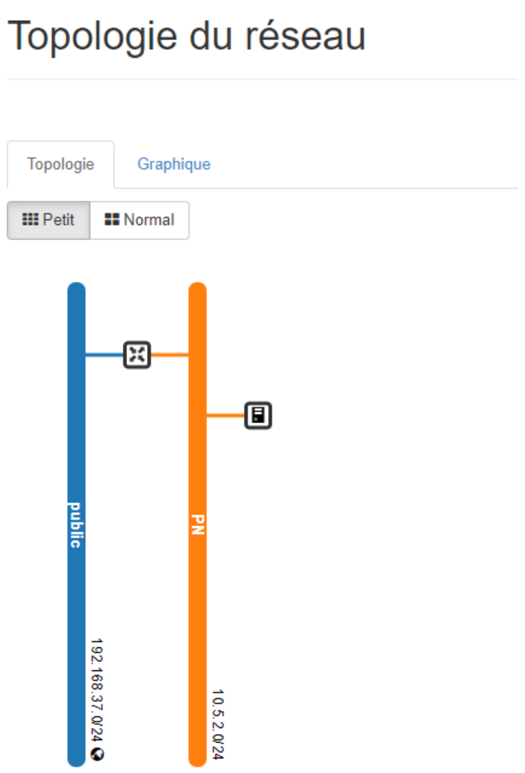
Openstack

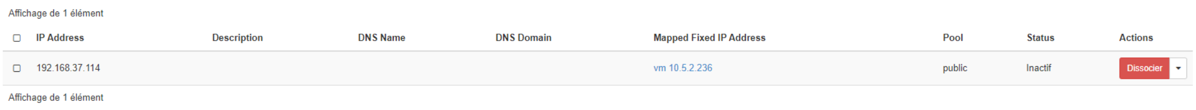
we created a private network, and a router to link it to the public insa network.
The VM IP address is 10.5.2.236. This is not a public address. hence when we ping our host machine from the VM, it only works one way.



```
--- 192.168.56.1 ping statistics ---
4 packets transmitted, 0 packets received, 100% packet loss
alpine-node:~#
```

To solve this issue, we must attribute a floating IP address to the VM that is in the form of the public insa network (192.168.37.___)

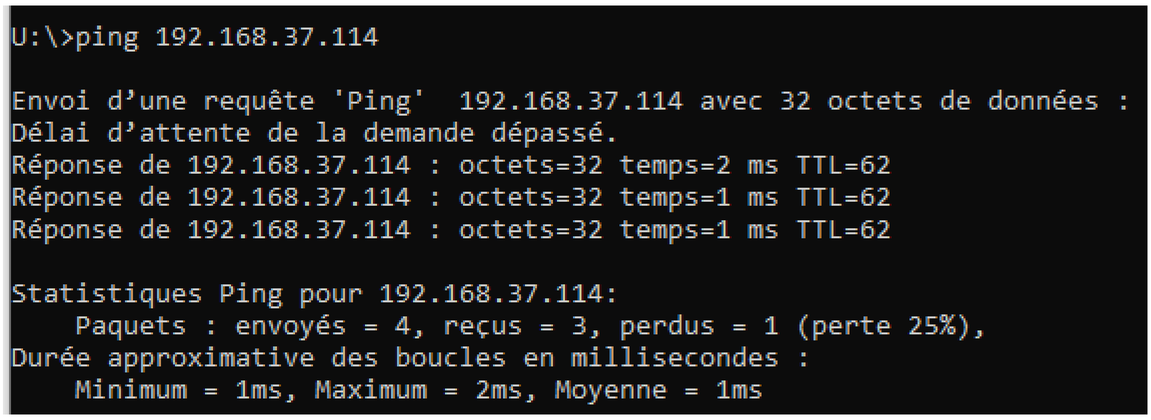




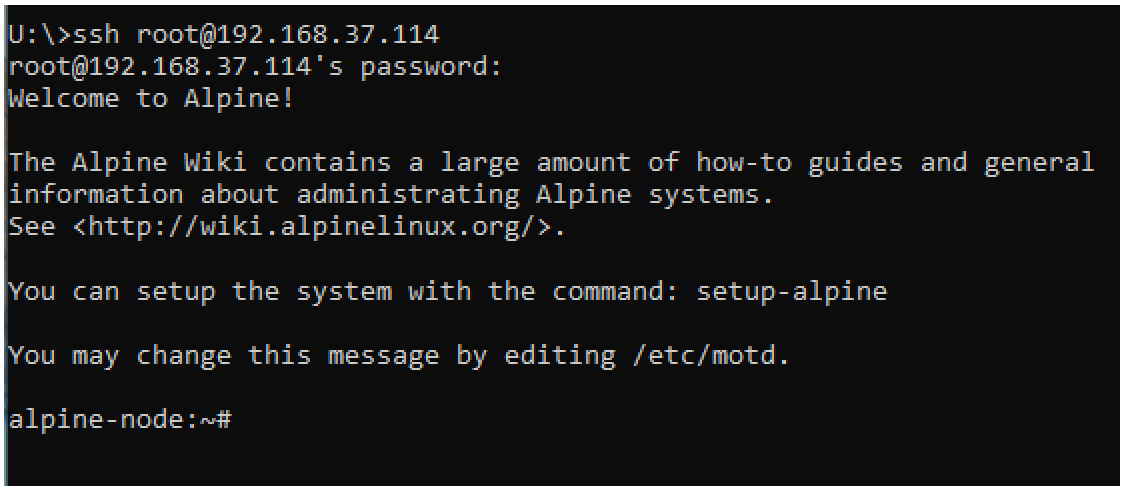
Affichage de 1 élément

☐	IP Address	Description	DNS Name	DNS Domain	Mapped Fixed IP Address	Pool	Status	Actions
☐	192.168.37.114				vm 10.5.2.236	public	Inactif	Dissocier

Affichage de 1 élément

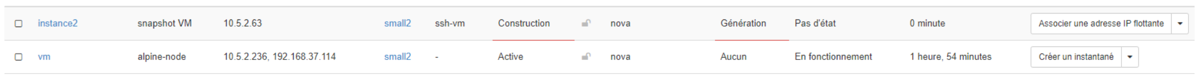


```
U:\>ping 192.168.37.114

Envoi d’une requête 'Ping'  192.168.37.114 avec 32 octets de données :
Délai d’attente de la demande dépassé.
Réponse de 192.168.37.114 : octets=32 temps=2 ms TTL=62
Réponse de 192.168.37.114 : octets=32 temps=1 ms TTL=62
Réponse de 192.168.37.114 : octets=32 temps=1 ms TTL=62

Statistiques Ping pour 192.168.37.114:
    Paquets : envoyés = 4, reçus = 3, perdus = 1 (perte 25%),
Durée approximative des boucles en millisecondes :
    Minimum = 1ms, Maximum = 2ms, Moyenne = 1ms
```



```
U:\>ssh root@192.168.37.114
root@192.168.37.114's password:
Welcome to Alpine!

The Alpine Wiki contains a large amount of how-to guides and general
information about administrating Alpine systems.
See <http://wiki.alpinelinux.org/>.

You can setup the system with the command: setup-alpine

You may change this message by editing /etc/motd.

alpine-node:~#
```

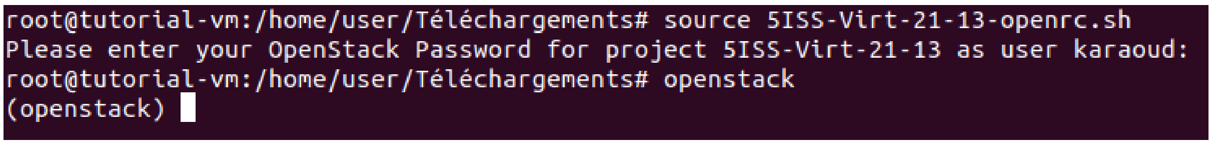
we notice that it's not possible to resize the running VM; we can only ask to ignore existing cores from what it already has access to. therefore we can somewhat downgrade it but not access larger options.



☐	instance2	snapshot VM	10.5.2.63	small2	ssh-vm	Construction	nova	Génération	Pas d'état	0 minute	Associer une adresse IP flottante
☐	vm	alpine-node	10.5.2.236, 192.168.37.114	small2	-	Active	nova	Aucun	En fonctionnement	1 heure, 54 minutes	Créer un instantané

Objectif 8 and 9:

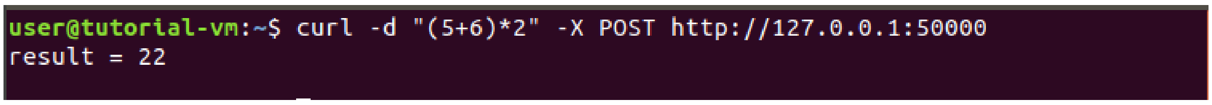
part1:
we install the openstack client on a VM (with VirtualBox and running Ubuntu), then configurate it with the RC v3 file downloaded from Openstack web GUI (Graphic User Interface). We can then start the client.



```
root@tutorial-vm:/home/user/Téléchargements# source 5ISS-Virt-21-13-openrc.sh
Please enter your OpenStack Password for project 5ISS-Virt-21-13 as user karaoud:
root@tutorial-vm:/home/user/Téléchargements# openstack
(openstack)
```

part2:
From there we deploy a tier-2 web application on OpenStack: the calculator. the four operations and the calculator service are implemented with Nodejs micro-services.

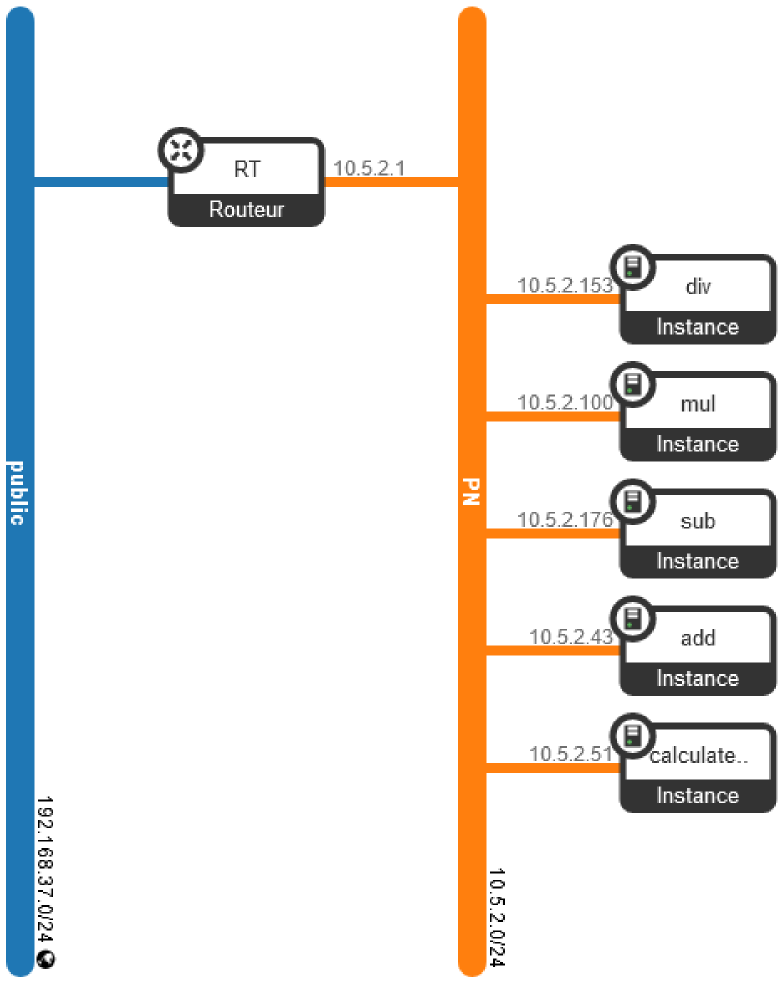
After installing all the required runtime for micro-services and making sure the calculatorservice is calling the right address and port for the right micro-services, we were able to make a simple calculation locally.



```
user@tutorial-vm:~$ curl -d "(5+6)*2" -X POST http://127.0.0.1:50000
result = 22
```

part3:
Here we create 5 VMs via the openstack web interface, we download on each of them one

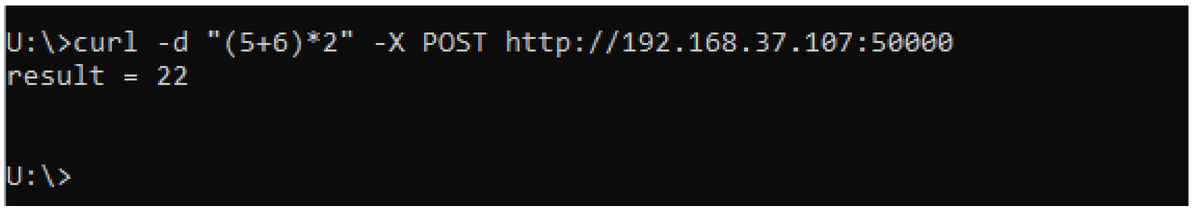
of the five micro-services necessary.



We give a remote IP to the VM containing the calculator service since it is linked to a private network and change the port it is listening to one that is allowed (50000). We also wrote the correct private IP addresses of the four micro-services VMs into the cs code so that it actually reach them via the private network

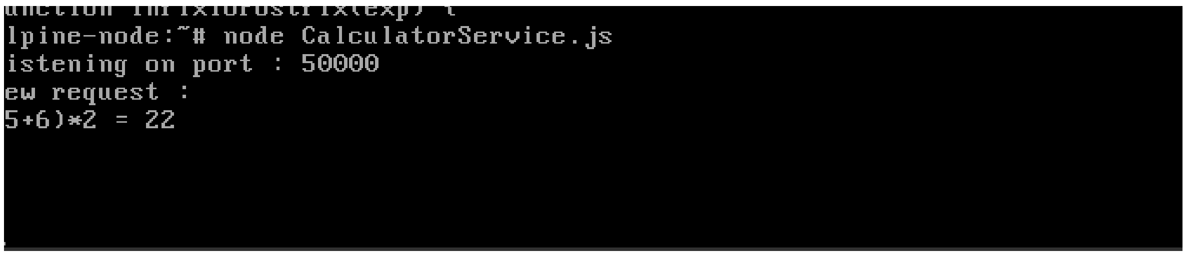
Finally, we changed the security groups in order to allow a http request (only on the port range 50000 to 50050) to reach the cs VM from our local host and for the cs VM to reach the four other micro-services's VMs.

Now we can send an http request from the host to the remote IP with a simple operation to get the result.
from the local host:



```
U:\>curl -d "(5+6)*2" -X POST http://192.168.37.107:50000
result = 22


U:\>
```

From the cs VM

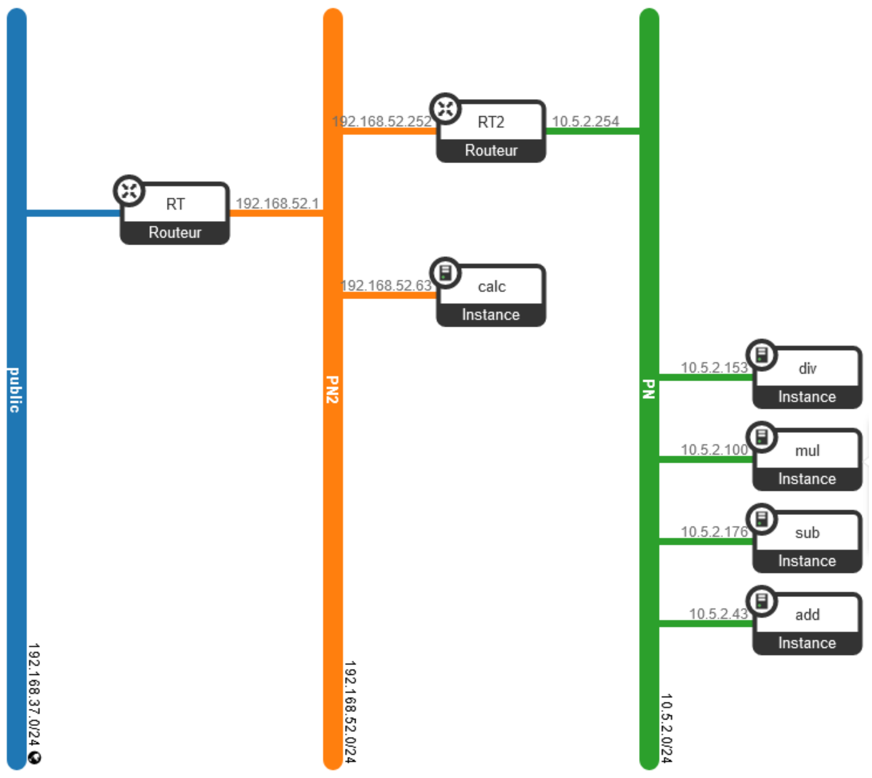


```
unction fnInfixToPostfix(exp) {
lpine-node:~# node CalculatorService.js
istening on port : 50000
ew request :
5+6)*2 = 22
```

For time reasons we will not focus on the fourth part regarding automation and orchestration of the application deployment.

objective 10 and 11:

For this part we want to have the micro-services and the cs on different networks, and only the network with the cs VM will be connected to the internet via the insa public network.
part two:



note: the intermediary network PN2 must not have the gateway option ON because there can only be one and it must be through the first router RT.

part three:
we cannot reach the microservices VMs from the cs VM.This is because the route between them through the second router does not exist.
we should add the route through the second router using the command “route add -net”.

Unfortunately we were not able to do so due to unknown issues regarding the routing.

part four:
we should use a docker API in nodejs to automate the creation of the structure.

we could also use the dockerfile where we identify all the VM required and all the actions necessary to create de structure. (from instancing the VMs, installing the runtimes