Groupe: ISS_A_1
Login 1: boukezza, Login 2: tberton
Nom prénom 1: Boukezzata Aymen, Nom prénom 2: Berton Thomas

Lab 1 : Introduction to Cloud Hypervisors

Theoretical part
(objectives 1 to 3)

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

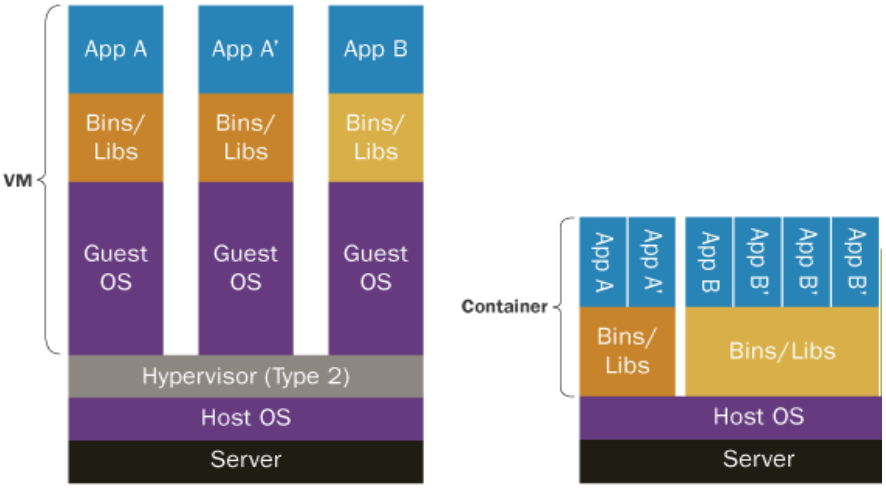


Figure 1: VM vs CT

- The figure represents the illustration of two different cloud infrastructures. The left one is based on virtual machines and the second one is based on containers.

  The virtual machine can be represented by running multiple operating systems (linux, windows, debian …) on one piece of hardware (the server). The Host OS (the main OS of the server), which handles the Hypervisor (like VMware Virtualbox ...) that manages the traffic of each guest OS,bins and Libs.

  The containers have the same hardware andey share the same Host Os. For each application, each container can install its own library and bins to execute the application.

	VM	CONTAINER
virtualization cost, taking into consideration memory size and CPU	The VM costs a high memory size (Go) for virtualization and consumes a large performance of the CPU	The containers take less memory size (Mo/Ko), because it needs only one host os.
Usage of CPU, memory and network for a given application	Each application needs its own guest os which use a large amount of cpu clock speed, memory and has different libraries.	For a given application, each container can share the same host os.
Security for the application (access right, resources sharing, etc.),	The applications are well isolated between each VM which make it more secure. It is isolated at the hardware level.	The isolation is on the OS level which make it less secure.
Performances (response time),	VM takes minutes to run, due to large size.	Container takes a few second to boot
Tooling for the continuous integration support	Each virtual machine has to be configured.	The configuration is easier because it shares the same os (libs and bins).

From an application developer point of view, the container is to be preferred because it is more performant and better optimized to configure.
From an infrastructure administrator's point of view, security is the most important, so the virtual machine is the best choice.

2. Similarities and differences between the existing CT types

Different CT technologies are available in the market (e.g. LXC/LXD, Docker, Rocket, OpenVZ, runC, containerd, systemd-nspawn). Their respective positioning is not obvious, but comparative analyses are available online, such as the one displayed in Figure 2.

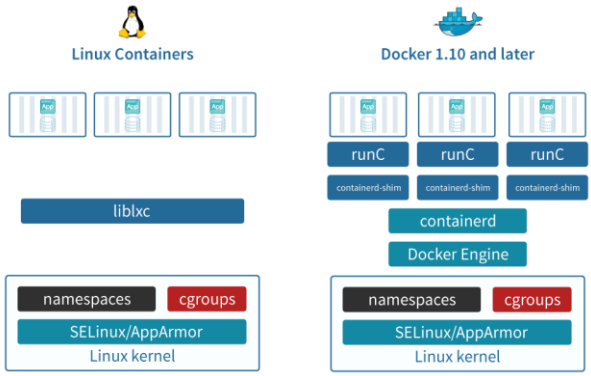


Figure 2 : Linux Lxc vs Docker

These technologies can however be compared based on the following criteria (non-exhaustive list):

- Application isolation and resources, from a multi-tenancy point of view,

Application isolation and resources is the separation of one program or application stack from the rest of the running application and multi-tenancy is an architecture in which a single instance of software application serves multiple customers.

- Containerization level (e.g. operating system, application),

Containerization level is the level of the shared resources between containers

- Tooling (e.g. API, continuous integration, or service composition).

Tooling are the set of services and applications that are available on each virtualization container technologies

	Application isolation and resources,	Containerization level	Tooling	multi-tenancy
Linux LXC	- Cgroup limits - Users limits - Shared network bridges	- privileged containers - Unprivileged containers	- Kernel namespaces (ipc, uts, mount, pid, network and user) - Apparmor and SELinux profiles - Seccomp policies - Chroots (using pivot_root) - Kernel	Yes, but is not a secure virtualization technology for multi-tenant environments

			capabilities - CGroups (control groups)	
Docker	- Kernel namespaces - Control groups - docker daemon attack surface	- Standard containers - Lightweight containers - Secure containers	- Easy and Faster Configuration - Swarm - Routing Mesh - Security Management	For multi-tenancy, docker would need to add a way to define users, and place them in a namespace

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

- There are two main categories of hypervisors, referred to as type 1 and type 2.

Type-1, native or bare-metal hypervisors

These hypervisors run directly on the host's hardware layer to manage guest operating systems.

Type-2 or hosted hypervisors

These hypervisors run on the operating system (OS) layer just as other computer programs do and guest operating system runs as a process on the host.

4. Difference between the two main network connection modes for virtualization hosts

With some hypervisors, multiple possibilities exist to connect a VM/CT to the Internet, via the host machine (in this case your desktop). The two main modes are the following:

- *NAT mode*: It is the default mode. It does not require any particular configuration. In this mode, the VM/CT is connected to a private IP network (i.e. a private address), and uses a virtual router (managed by the hypervisor) running on the host machine to communicate outside of the network:
    - This router executes what is equivalent to a NAT function (only *postrouting*) allowing the VM to reach the host machine or the outside;
    - However, the VM cannot be reached from the host (and by any another VM hosted on the same machine): to make it accessible from outside, it is necessary to deploy port forwarding (*prerouting*).
- *Bridge mode*, the most widely used (yet not systematically), in which the VM/CT sees itself virtually connected to the local network of its host (see Figure 3): it has an IP address identifying it on the host network, and can access the Internet (or is accessed) as the host.

NB: Other less used modes exist, such as Private Network in VirtualBox, that you can try out.

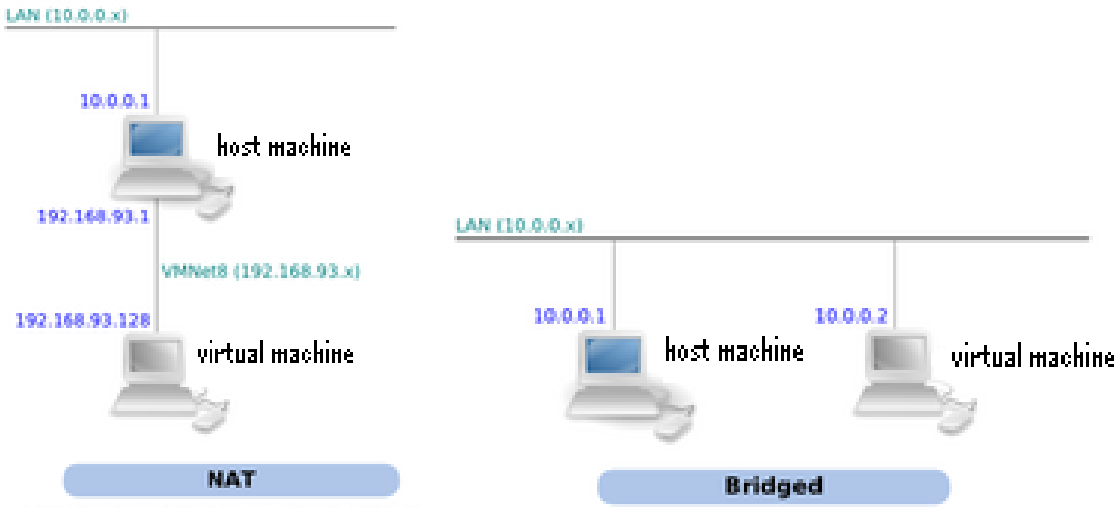


Figure : Mode NAT vs Mode Bridge (the most usual)

Practical part
(objectives 4 to 7)

For each item thereafter, you must make a short written report on the shared document (with snapshots when applicable).

1. Tasks related to objectives 4 and 5

Creating a VirtualBox VM (in NAT mode), and setting up the network to enable two-way communication with the outside

In this part, you will use the VirtualBox hypervisor in NAT mode to connect a VM to the network. The bridge mode will be used in the second part of the lab, with another hypervisor (OpenStack).

First part: Creating and configuring a VM

We configured and created the VM

Second part: Testing the VM connectivity

- Identify the IP address attributed to the VM using ifconfig (in Linux command line), and compare it to the host address (ip config in the command line). What do you observe?

We observe that the IP address of the host is different from the IP address of the VM
IP address of the host : 10.0.5.234
IP address of the VM : 10.0.2.15

With a ping command (or any other command of your choice) :

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

The connectivity from the VM to the outside worked, because the router used NAT features that enable the VM to act much like a real computer that connects to the Internet through a router

- Check the connectivity from your neighbour's host to your hosted VM. What do you observe?

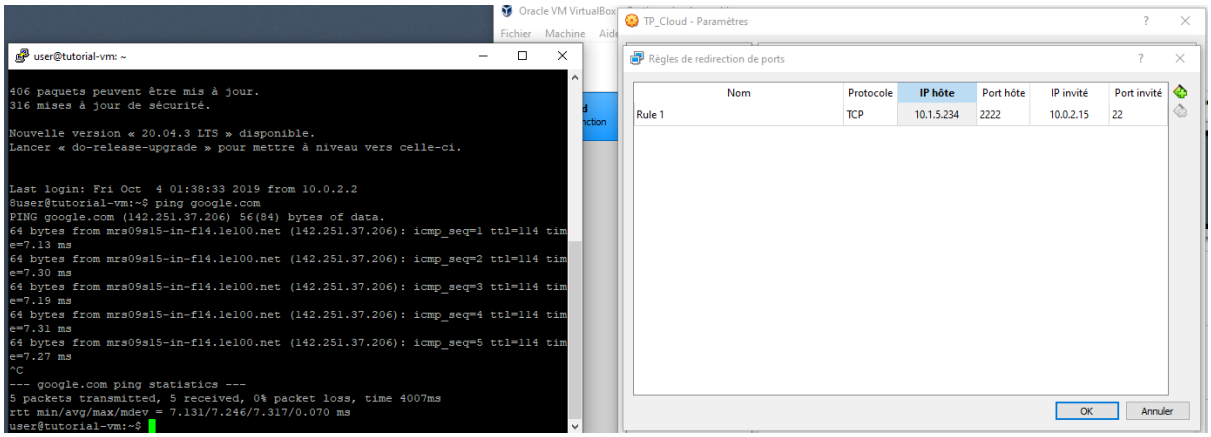
The ping command couldn't access the neighbour's hots. similarly from the side of the VM

- Check the connectivity from your host to the hosted VM. What do you observe?

Same result as the connectivity to the neighbour's. The can connect to the outside using the NAT router but not the opposite. So we have to configure a port forwarding because a VM can not virtualize a network card.

Third part: Set up the "missing" connectivity

To be able to connect to the virtual machine, we use the port-forwarding method. For that, it is necessary to configure the virtual machine starting from the supervisor where one must indicate the IP address of the host (the machine) and the guest (the VM) and the ports of each one.
When we send a packet to the IP address and the host port it will be transmitted directly to the virtual machine, it is the principle of port-forwarding.
(We used Putty SSH to connect the VM from the host)



Fourth part: VM duplication

We create a new clone of the VM disk file with the following commande:

```
VBoxManage clonemedium "C:\user\boukezza\Downloads\disk.vmdk\disk.vmdk "C:\user\boukezza\Dowloads\disk.copy.vmdk
```

Then we create the VM with the new file disk "disk.copy.vmdk"

Fifth part: Docker containers provisioning

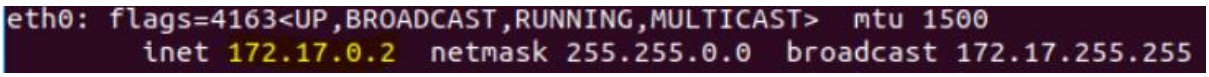
We have got first an ubuntu image and executed a first container instance CT1 of the ubuntu image with Docker and then we have installed all the required connectivity tools “net-tool iputils-ping”

```
$ sudo docker pull ubuntu
```

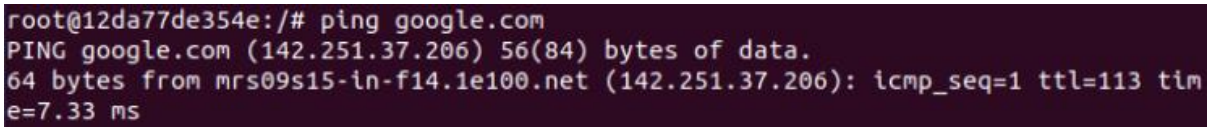
```
$ sudo docker run --name ct1 -it ubuntu
```

```
$ sudo apt-get -y update && apt-get -y install net-tools iputils-ping
```

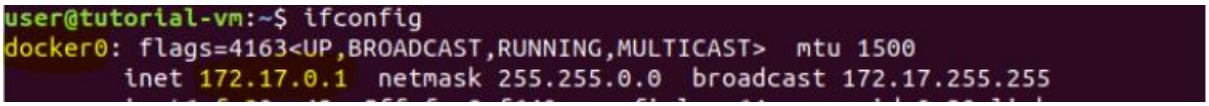
4. Check the connectivity (through ping) with the newly instantiated Docker:
    1. What is the Docker IP address?: 172.17.0.2



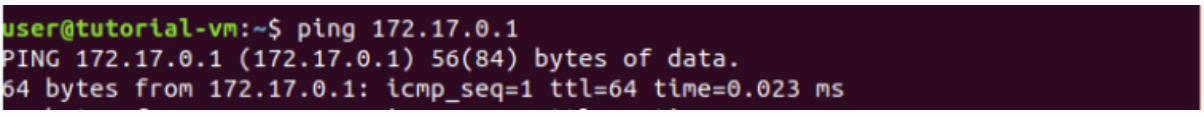
    2. Ping an Internet resource from Docker OK



    3. Ping the VM from Docker OK



    4. Ping the Docker from the VM



    Elaborate on the obtained results

As the docker is created from the VM, it is normal that we can establish a connection between each other. Moreover the docked IP address has been added to the routing table.

After that we executed a new container instance CT2 of the ubuntu docker and installed the tet editor “nano”

```
$ sudo docker run --name ct2 -p 2223:22 -it ubuntu
```

```
[CT2] $ apt-get -y update && apt install nano
```

To make a snapshot of a snapshot of the CT2

```
user@tutorial-vm:~$ sudo docker ps
CONTAINER ID        IMAGE               COMMAND             CREATED             STATUS
 PORTS                   NAMES
4334447d9a3b        ubuntu              "bash"              8 minutes ago       Up 8 minutes
 0.0.0.0:2223->22/tcp   ct2
user@tutorial-vm:~$ sudo docker images
REPOSITORY          TAG                 IMAGE ID            CREATED             SIZE
ubuntu              latest              597ce1600cf4        4 days ago          72.8MB
user@tutorial-vm:~$ sudo docker commit 4334447d9a3b ubuntu:latest
sha256:960ae7245d5e6eb02750238bee4a439b8cbea15b5772084af2bba3554b728276
```

To stop the CT

```
user@tutorial-vm:~$ sudo docker stop 4334447d9a3b
4334447d9a3b
```

To terminate the CT

```
user@tutorial-vm:~$ sudo docker rm 4334447d9a3b
4334447d9a3b
```

List the available Docker images in the VM

```
user@tutorial-vm:~$ sudo docker images
REPOSITORY          TAG                 IMAGE ID            CREATED
SIZE
ubuntu              latest              960ae7245d5e        4 minutes ago
105MB
ubuntu              <none>              597ce1600cf4        4 days ago
72.8MB
user@tutorial-vm:~$
```

10. Execute a new instance (CT3) from the snapshot that you previously created from CT2 using the run command. Do you still have nano installed on CT3? Explain the reason in the shared document.

We created CT2 and installed nano, a snapshot of the CT2 used to create CT3 will also have nano installed

11. Docker enables “recipes” sharing to create persistent images (as a second alternatives of snapshots). To make a proper recipe, you need to write a *Dockerfile (myDocker.dockerfile)*:

```
FROM ubuntu

RUN apt update -y

RUN apt install -y nano

CMD ["/bin/bash"]
```

```
user@tutorial-vm:~$ sudo docker build -t ubuntu:latest -f mydocker.dockerfile .
Sending build context to Docker daemon  754.2kB
Step 1/4 : FROM ubuntu
 ---> 960ae7245d5e
Step 2/4 : RUN apt update -y
 ---> Running in a2c9bc79f092
Hit:1 http://archive.ubuntu.com/ubuntu focal InRelease
Get:2 http://archive.ubuntu.com/ubuntu focal-updates InRelease [114 kB]
Get:3 http://archive.ubuntu.com/ubuntu focal-backports InRelease [101 kB]
Get:4 http://security.ubuntu.com/ubuntu focal-security InRelease [114 kB]
Fetched 328 kB in 1s (327 kB/s)
Reading package lists... Done
Building dependency tree
Reading state information... Done
All packages are up to date.
Removing intermediate container a2c9bc79f092
 ---> 07e28ba1aded
Step 3/4 : RUN apt install -y nano
 ---> Running in e66fcc0e2853
Reading package lists... Done
Building dependency tree
Reading state information... Done
nano is already the newest version (4.8-1ubuntu1).
0 upgraded, 0 newly installed, 0 to remove and 0 not upgraded.
Removing intermediate container e66fcc0e2853
 ---> fa4599669ab4
Step 4/4 : CMD ["/bin/bash"]
 ---> Running in 6ad91686ea16
Removing intermediate container 6ad91686ea16
 ---> 3cc33f57102f
Successfully built 3cc33f57102f
Successfully tagged ubuntu:latest
```

2. Expected work for objectives 6 and 7

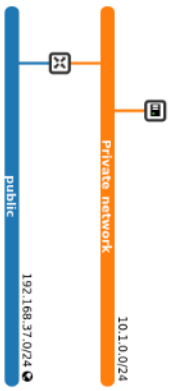
First part : CT creation and configuration on OpenStack

After that we got connected to the Openstack Web interface, we created a private network 10.4.5.0/24 then we associated an instance VM to this network

Second part: Connectivity test

The VM IP address that was given by the hypervisor is displayed on the dashboard (Instances tab). What do you think about this address? Write down your comments on the shared document.

The VM IP address that was given by the hypervisor 10.4.5.108 belongs to the private network connected to it.



- Using a Ping (or any other appropriate command) :
    - Check the connectivity from the VM to the desktop. Write down your comments

The connectivity is OK

    - Check the connectivity from the desktop to the VM. Write down your comments.

We couldn't connect to the VM from the desktop, because Openstack create virtual local network and the VM's IP are not routable. So we have to allocate to the VM a floating IP address

Instance name	Image name	IP address	Flavor	Key pair	Status	Availibility zone	Task	Power state
VM	ubuntu4CLV	10.4.5.108 192.168.37.163	small2	-	Active	nova	Aucun	En fonctionnement

Third part: Snapshot, restore and resize a VM

- Resize a running VM from the Instances tab. What do you observe? Write down your comments.

we resized a running VM. If the selected size is under the VM instance size created and too small, the resizing doesn't work. otherwise if it is big enough the operation will succeed.

- Shutdown the VM and redo the same operation. Write down your comments.

If we try again with shutdown VM's the operation succeeds

- Make a snapshot of the VM. Elaborate on the differences between the included material/software within the original image and the newly created snapshot

We took a snapshot of the VM. We compared it to the included material/software.

	Type	Visibility	Disk format	Size
ubuntu4CLV	Image	Public	QCOW2	3,66 Go
Snapshot	Snapshot	Private	QCOW2	3,78 Go

3. Expected work for objectives 8 and 9

Part one : OpenStack client installation

We installed the openstack REST client and configured it with the RC file downloaded from the openstack web account.

Part two : Web 2-tier application topology and specification

Sum service

```
user@tutorial-vm:~$ curl -d "10 13" -X POST http://localhost:50001
23
user@tutorial-vm:~$

Listening on port : 50001
New request :
A = 10
B = 13
A + B = 23
```

Subtraction service

```
user@tutorial-vm:~$ curl -d "10 13" -X POST http://localhost:50002
-3
user@tutorial-vm:~$
^C
user@tutorial-vm:~/cloud$ node SubService.js
Listening on port : 50002
New request :
A = 10
B = 13
A - B = -3
```

Division service

```
user@tutorial-vm:~$ curl -d "13 10" -X POST http://localhost:50004
1.3
user@tutorial-vm:~$
^C
user@tutorial-vm:~/cloud$ node DivService.js
Listening on port : 50004
New request :
A = 13
B = 10
A / B = 1.3
```

Multiplication service

```
user@tutorial-vm:~$ curl -d "13 10" -X POST http://localhost:50003
130
user@tutorial-vm:~$
^C
user@tutorial-vm:~/cloud$ node MulService.js
Listening on port : 50003
New request :
A = 13
B = 10
A * B = 130
```

Calculator Service

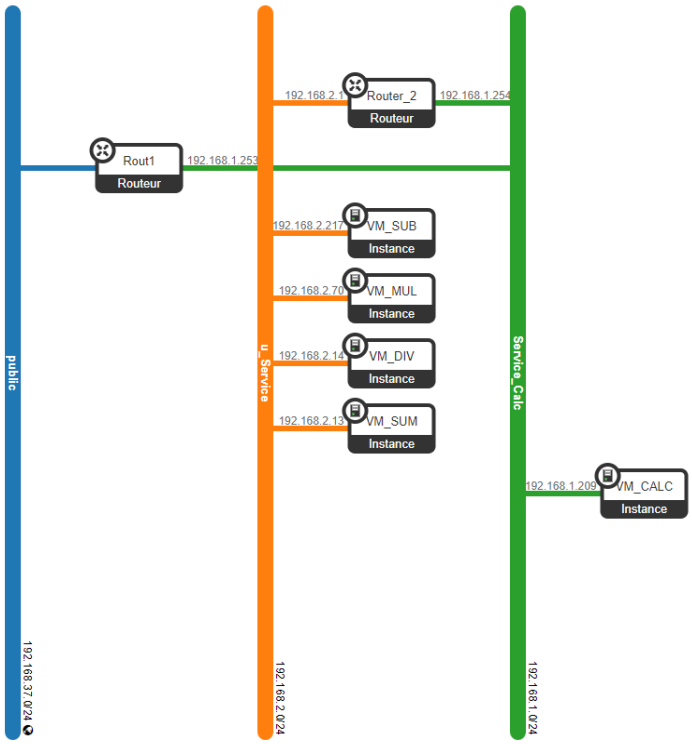
```
user@tutorial-vm:~/cloud$ emacs CalculatorService.js
^[[A^[[A^Cuser@tutorial-vm:~/cloud$ node CalculatorService.js
Listening on port : 50000
New request :
(5+3)*9 = 72


user@tutorial-vm:~$ curl -d "(5+3)*9" -X POST http://localhost:50000
result = 72
```

The calculator service call all the other 4 services

4. Expected work for objectives 10 and 11

Part one: The client requirements and target network topology



Part two: Deployment of the target topology

Since VM_CALC is connected to the internet, we installed NodeJS and downloaded all the services with WGET

Part three: Configuration of the topology and execution of the services

- Test the connectivity between the VM hosting the calculator front end and any machine from the ones hosting the arithmetic operations services. What do you observe?

Elaborate on this?

We couldn't get connected to micro services from the VM calculator, simply because the arithmetic operations services network is not included in the VM_Calc route table.

We added the micro services network and the gateway to the VM_CALC network table.

```
user@tutorial-vm:~$ sudo route add -net 30.4.5.0/24 gw 20.4.5.4
user@tutorial-vm:~$ route
Table de routage IP du noyau
Destination     Passerelle      Genmask         Indic Metric Ref    Use Iface
default         host-20-4-5-1.l 0.0.0.0         UG    100    0        0 ens3
20.4.5.0        0.0.0.0         255.255.255.0   U     0      0        0 ens3
30.4.5.0        host-20-4-5-4.l 255.255.255.0   UG    0      0        0 ens3
169.254.169.254 host-20-4-5-1.l 255.255.255.255 UGH   100    0        0 ens3
user@tutorial-vm:~$
```

After that we succeed to ping the VM's from both sides

```
user@tutorial-vm:~$ ping 30.4.5.218
PING 30.4.5.218 (30.4.5.218) 56(84) bytes of data.
64 bytes from 30.4.5.218: icmp_seq=1 ttl=64 time=3.43 ms
64 bytes from 30.4.5.218: icmp_seq=2 ttl=64 time=0.809 ms
^C
--- 30.4.5.218 ping statistics ---
2 packets transmitted, 2 received, 0% packet loss, time 1001ms
rtt min/avg/max/mdev = 0.809/2.123/3.438/1.315 ms
user@tutorial-vm:~$ ping 30.4.5.54
PING 30.4.5.54 (30.4.5.54) 56(84) bytes of data.
64 bytes from 30.4.5.54: icmp_seq=1 ttl=64 time=1.56 ms
64 bytes from 30.4.5.54: icmp_seq=2 ttl=64 time=1.06 ms
^C
--- 30.4.5.54 ping statistics ---
2 packets transmitted, 2 received, 0% packet loss, time 1002ms
rtt min/avg/max/mdev = 1.061/1.315/1.569/0.254 ms
user@tutorial-vm:~$ ping 30.4.5.120
PING 30.4.5.120 (30.4.5.120) 56(84) bytes of data.
64 bytes from 30.4.5.120: icmp_seq=1 ttl=64 time=2.35 ms
^C
--- 30.4.5.120 ping statistics ---
1 packets transmitted, 1 received, 0% packet loss, time 0ms
rtt min/avg/max/mdev = 2.358/2.358/2.358/0.000 ms
user@tutorial-vm:~$
```

```
user@tutorial-vm:~$ ping 20.4.5.88
PING 20.4.5.88 (20.4.5.88) 56(84) bytes of data.
^C
--- 20.4.5.88 ping statistics ---
6 packets transmitted, 0 received, 100% packet loss, time 5126ms
```

We executed all the services in each VM and edited the CalculatorService.js with the right micro services IP addresses
(Since the micro services VM didn't have internet connexion and VM_CALC did we downloaded all the services and NodeJS on VM_CALC then we transferred each one of them with SFTP and installed NodeJS manually)

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

const PORT = process.env.PORT || 50000;

const SUM_SERVICE_IP_PORT = 'http://30.4.5.215:50001';
const SUB_SERVICE_IP_PORT = 'http://30.4.5.109:50002';
const MUL_SERVICE_IP_PORT = 'http://30.4.5.120:50003';
const DIV_SERVICE_IP_PORT = 'http://30.4.5.215:50004';



String.prototype.isNumeric = function() {
    return !isNaN(parseFloat(this)) && isFinite(this);
}
Array.prototype.clean = function() {
    for(var i = 0; i < this.length; i++) {
        if(this[i] === "") {
            this.splice(i, 1);
        }
    }
    return this;
}
```

We tested the Calculator post request and it worked !!!