5ISS - Cloud Computing: Adaptability and Autonomic Management

Lab 1 : Introduction to Cloud Hypervisors

Supervisor:
S. Yangui (INSA/LAAS)

Students:
ESTIVAL Emilie, PAGÈS Maxime

Introduction

In this lab we learned about virtualization techniques. These techniques are used in IT environments that support the provisioning (i.e. development, deployment, management) of novel software systems (with their potential constraints such as QoS and security). These environments are typically dynamic and distributed.

Objectives

In this part we will outline the objectives of the lab:

- **Objective 1**: Recognize the fundamental differences between the types of hypervisors' architectures (type 1/type 2).
- **Objective 2**: Recognize the fundamental differences between the two main types of *virtualisation hosts*, i.e. *virtual machines* (VM) and *containers* (CT).
- **Objective 3**: Evaluate the two different modes of network connection for VM and CT (i.e. NAT and Bridge modes).
- **Objective 4**: Operate VirtualBox VMs provisioning, in NAT mode, and make appropriate configurations so that VMs can access and be accessed through the Internet
- **Objective 5**: Operate Docker containers provisioning, and make appropriate configurations so that containers can access and be accessed through the Internet
- **Objective 6**: Operate proper VMs provisioning with OpenStack and manage the networking connectivity
- **Objective 7**: Test and evaluate the supported management operations on the virtualization hosts (e.g. snapshot, restore, backup, resize).
- **Objective 8**: Use the OpenStack API in order to automate the operations described in objectives 4, 5, 6 and 7.
- **Objective 9**: Implement a simple orchestration tool in order to provision a Web 2-tier application.
- **Objective 10** : Make up a specific network topology on OpenStack
- **Objective 11**: Configure and deploy this network topology

Theoretical part

(objectives 1 to 3)

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

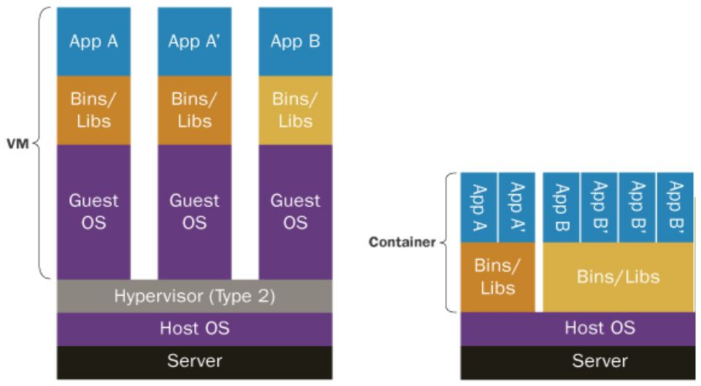


Figure 1: VM vs CT

We can define a VM as a *virtual server that emulates a hardware server*. The official definition from the VMware website[1] is the following: *"A virtual machine is a software computer that, like a physical computer, runs an operating system and applications"*. But VMs provide an abstraction layer and hide the complexities of underlying hardware components. From Figure 1, we can see that a VM relies on the system's physical hardware to emulate a similar environment on which you can install applications. To do so, a VM is based on a hypervisor which is a hardware virtualization technique that allows multiple (gest) OS to run on a single host system at the same time. The guest OS are all sharing the host OS. Each guest OS is launched within a separate virtual machine. The VMs thus have their own kernel, drivers and the binaries of the applications.

As a result, if an application wants to interact with a CPU, it will have to pass through the operating system and then the hypervisor. We can imagine that passing through hypervisor and host OS can *impact the user experience*. VMs introduce problems because they are memory-hungry programs, and also add *duplication of application binaries between different virtual machines*. Additionally, it takes a longer time to boot up a VM.

While virtual machines *clone an operating system for themselves*, *containers can share an operating system*. We can see from Figure 1 that containers do not require a guest OS. Unlike a VM, a container allows applications to communicate with the CPU without passing through the guest operating system and then the

[1] Link, Link2

hypervisor. As a result, we can imagine that containers offer better performance than VMs.

Containers allow an *isolation between processes which appear as separate OS*. This enables applications to run in a secure environment. The environment variables along with the libraries are stored within the containers. This promotes faster execution of instructions in containers when compared to the hypervisor based instances like VMs.

We summed up the main differences between VMs and containers in a comparative table according to given criteria. For each criterion, we also precised the implications regarding application developer / infrastructure administrator depending for whom it was the most relevant.

Criteria	**Virtual machine**	**Container**	**Application developer or infrastructure administrator**
Virtualization cost, taking into consideration memory size and CPU	Runs a complete operating system including the kernel, thus requiring more system resources (CPU, memory, and storage).	Runs the user mode portion of an operating system, and can be tailored to contain just the needed services for your app, using fewer system resources.	Infrastructure administrators would prefer to use a VM.
Usage of CPU, memory and network for a given application	Limited performance: there is a hardware emulation.	Native performance: a containerized application processes run in an isolated environment directly on the host OS.	Application developers would prefer to use a container.
Security for the application (access right, resources sharing, etc.)	A VM provides a strong isolation between the Host OS and other VMs.	Lightweight isolation between the Host OS and other containers.	Infrastructure administrators would prefer to use a VM.
Performances (response time)	Startup time in minutes. High response time.	Startup time in milliseconds. Low response time(depending of the type of workload you are contending with)	Application developers would prefer to use a container.
Tooling for the continuous integration support	Installing a new operating system version requires upgrading or often just creating an entirely new VM. This can be time-consuming.	Excellent for Continuous Integration and Continuous Deployment.	Infrastructure administrators would prefer to use a container.

2. Similarities and differences between the existing CT types

First, we decided to draw the pros and cons of the container technology, which we summed up in the following table.

	CONTAINERS	
1	PROS	CONS
2	Application level isolation	Not as mature as VMs, their performance is still to be tested on large scales
3	Faster to set up than a VM and takes less memory	Difficult to support the applications in long-term because not many IT admin have experience with container technology yet
4	Easier to migrate, back up and transport due to smaller sizes compared to VM	It adds another tool to manage in the IT infrastructure
5	Faster communication to hardware, efficient for performance	As the applications are not fully isolated, they are not as secure as they are in VMs
6	Improves application deployment and maintenance due to self-contained container images	
7	Reduces time to deliver the application from the developer's point of view	

But different CT technologies are available in the market (e.g. LXC/LXD, Docker, Rocket, OpenVZ, runC, containerd, systemd-nspawn). In the following table, we compare three of these technologies based on the following criteria. [2]

Criteria	Linux Lxc	Docker	Rocket
Application isolation and resources	LXC containers can execute multiple applications and processes.	Docker containers are restricted to a single application or a service.	No high-privilege daemons are required for RKT to work properly, it is not necessarily necessary to have administrator privileges
Containerization level	LXC is a container technology that gives us the lightweight Linux containers.	Docker is single application virtualization which is based on top of the containers.	Same as Docker but can also support other formats.
Data persistence	LXC containers are complete virtualization	In a Docker container, changes made to the	Data is always retrievable.

[2] https://www.educba.com/lxc-vs-docker/
https://www.educba.com/rkt-vs-docker/

	entities with its own file system. So any data updated in an LXC container, will always be retrievable.	data cannot persist beyond a restart.	

Application developer or infrastructure administrator?

An infrastructure administrator would prefer LXC containers because LXC containers can be used to host virtual environments for private hosting, just like how VMs are used. Meanwhile, an application developer would prefer Docker.

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: Bare-metal (a.k.a Native or Hardware-level): hypervisor runs directly on the system hardware. With direct access to the underlying hardware and no other software -- such as OSes and device drivers -- to contend with, Type 1 hypervisors are regarded as the most efficient and best-performing hypervisors available for enterprise computing. Hypervisors such as VMware ESXi, Microsoft Hyper-V server and open source KVM are examples of Type 1 hypervisors.[3]

Type 2: Hosted (a.k.a OS-level) - Virtual Box: runs on a host operating system that provides virtualization services, such as I/O device support and memory management. It is called a hosted hypervisor because it relies on the host machine's pre-existing OS to manage calls to CPU, memory, storage and network resources. Type 2 hypervisors include VMware Fusion, Oracle VM VirtualBox, Oracle VM Server for x86, Oracle Solaris Zones, Parallels and VMware Workstation.

[3] Link, Link

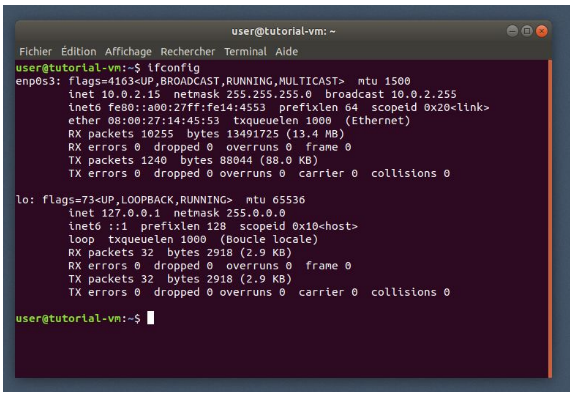
Practical part
(objectives 4 to 7)

1. Tasks related to objectives 4 and 5

In this part, we 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
First of all, we create and configure a VM with VirtualBox. It is a Linux 64 bits VM with a NAT mode network board. The private IP address is automatically attributed by VirtualBox.

Second part: Testing the VM connectivity
In this part, we want to test the connectivity of the VM. From the VM and thanks to the "ifconfig" command line, we identify the IP address attributed to the VM as shown in the figure below.



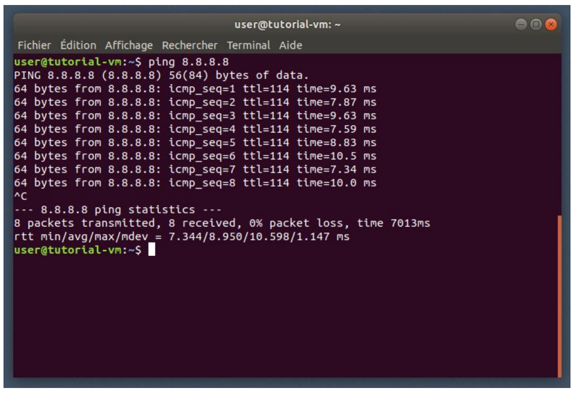
```
user@tutorial-vm:~$ ifconfig
enp0s3: flags=4163<UP,BROADCAST,RUNNING,MULTICAST>  mtu 1500
        inet 10.0.2.15  netmask 255.255.255.0  broadcast 10.0.2.255
        inet6 fe80::a00:27ff:fe14:4553  prefixlen 64  scopeid 0x20<link>
        ether 08:00:27:14:45:53  txqueuelen 1000  (Ethernet)
        RX packets 10255  bytes 13491725 (13.4 MB)
        RX errors 0  dropped 0  overruns 0  frame 0
        TX packets 1240  bytes 88044 (88.0 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 32  bytes 2918 (2.9 KB)
        RX errors 0  dropped 0  overruns 0  frame 0
        TX packets 32  bytes 2918 (2.9 KB)
        TX errors 0  dropped 0 overruns 0  carrier 0  collisions 0

user@tutorial-vm:~$ 
```

VM IP address : **10.0.2.15**.
Host IP address : **10.1.5.22**.

The IP address of the VM is different from the IP address of the host. That's because we configured the VM with a NAT mode network board.

Now we test the connectivity of the VM with the outside, with a neighbour's host and with the host using the ping command.

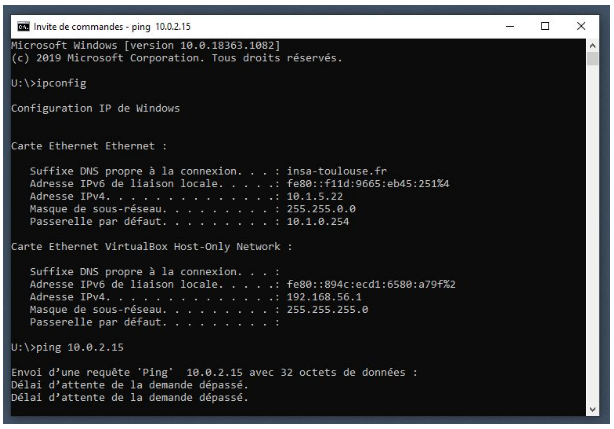
- Connectivity from the VM to the outside:



```
user@tutorial-vm:~$ 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=114 time=9.63 ms
64 bytes from 8.8.8.8: icmp_seq=2 ttl=114 time=7.87 ms
64 bytes from 8.8.8.8: icmp_seq=3 ttl=114 time=9.63 ms
64 bytes from 8.8.8.8: icmp_seq=4 ttl=114 time=7.59 ms
64 bytes from 8.8.8.8: icmp_seq=5 ttl=114 time=8.83 ms
64 bytes from 8.8.8.8: icmp_seq=6 ttl=114 time=10.5 ms
64 bytes from 8.8.8.8: icmp_seq=7 ttl=114 time=7.34 ms
64 bytes from 8.8.8.8: icmp_seq=8 ttl=114 time=10.0 ms
^C
--- 8.8.8.8 ping statistics ---
8 packets transmitted, 8 received, 0% packet loss, time 7013ms
rtt min/avg/max/mdev = 7.344/8.950/10.598/1.147 ms
user@tutorial-vm:~$
```

As we can see in this figure, the connectivity is effectively provided through the NAT mode network.

- Connectivity from the neighbour's host to your hosted VM:

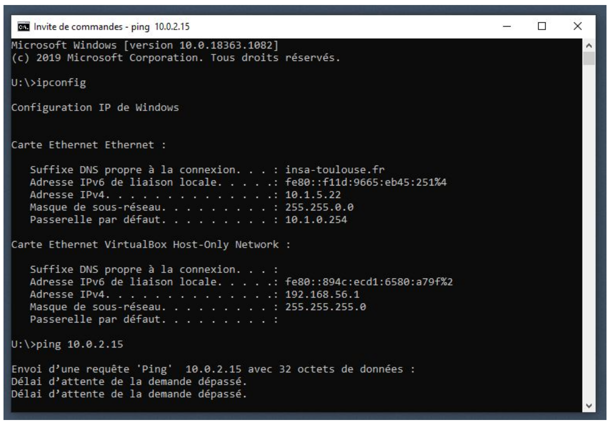


```
Microsoft Windows [version 10.0.18363.1082]
(c) 2019 Microsoft Corporation. Tous droits réservés.

U:\>ipconfig

Configuration IP de Windows


Carte Ethernet Ethernet :

   Suffixe DNS propre à la connexion. . . : insa-toulouse.fr
   Adresse IPv6 de liaison locale. . . . .: fe80::f11d:9665:eb45:251%4
   Adresse IPv4. . . . . . . . . . . . . .: 10.1.5.22
   Masque de sous-réseau. . . . . . . . . : 255.255.0.0
   Passerelle par défaut. . . . . . . . . : 10.1.0.254

Carte Ethernet VirtualBox Host-Only Network :

   Suffixe DNS propre à la connexion. . . :
   Adresse IPv6 de liaison locale. . . . .: fe80::894c:ecd1:6580:a79f%2
   Adresse IPv4. . . . . . . . . . . . . .: 192.168.56.1
   Masque de sous-réseau. . . . . . . . . : 255.255.255.0
   Passerelle par défaut. . . . . . . . . :

U:\>ping 10.0.2.15

Envoi d’une requête 'Ping'  10.0.2.15 avec 32 octets de données :
Délai d’attente de la demande dépassé.
Délai d’attente de la demande dépassé.
```

During a ping, the timeout period is exceeded. It is impossible to connect to the neighboring machine.

The VM IP address is not routable, i.e. it is not defined on the network. It is therefore impossible to reach the machine from the outside.

- Connectivity from the host to the hosted VM:



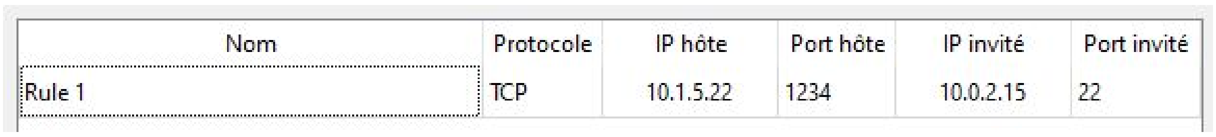
```
Microsoft Windows [version 10.0.18363.1082]
(c) 2019 Microsoft Corporation. Tous droits réservés.

U:\>ipconfig

Configuration IP de Windows


Carte Ethernet Ethernet :

   Suffixe DNS propre à la connexion. . . : insa-toulouse.fr
   Adresse IPv6 de liaison locale. . . . .: fe80::f11d:9665:eb45:251%4
   Adresse IPv4. . . . . . . . . . . . . .: 10.1.5.22
   Masque de sous-réseau. . . . . . . . . : 255.255.0.0
   Passerelle par défaut. . . . . . . . . : 10.1.0.254

Carte Ethernet VirtualBox Host-Only Network :

   Suffixe DNS propre à la connexion. . . :
   Adresse IPv6 de liaison locale. . . . .: fe80::894c:ecd1:6580:a79f%2
   Adresse IPv4. . . . . . . . . . . . . .: 192.168.56.1
   Masque de sous-réseau. . . . . . . . . : 255.255.255.0
   Passerelle par défaut. . . . . . . . . :

U:\>ping 10.0.2.15

Envoi d’une requête 'Ping'  10.0.2.15 avec 32 octets de données :
Délai d’attente de la demande dépassé.
Délai d’attente de la demande dépassé.
```

Same observation here, the VM IP address is not routable. The connectivity from the host to the hosted VM is not ensured.

Third part: Set up the "missing" connectivity

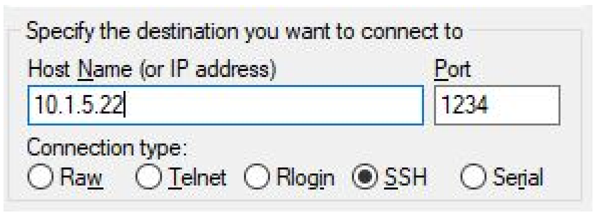
In this part, we propose to fix the issues identified in the previous part, for a dedicated application (e.g. SSH, port 22). To that end, we are going to use the Port Forwarding technique.

As shown below, we add a new forwarding rule to the management interface of the network board provided by VirtualBox.



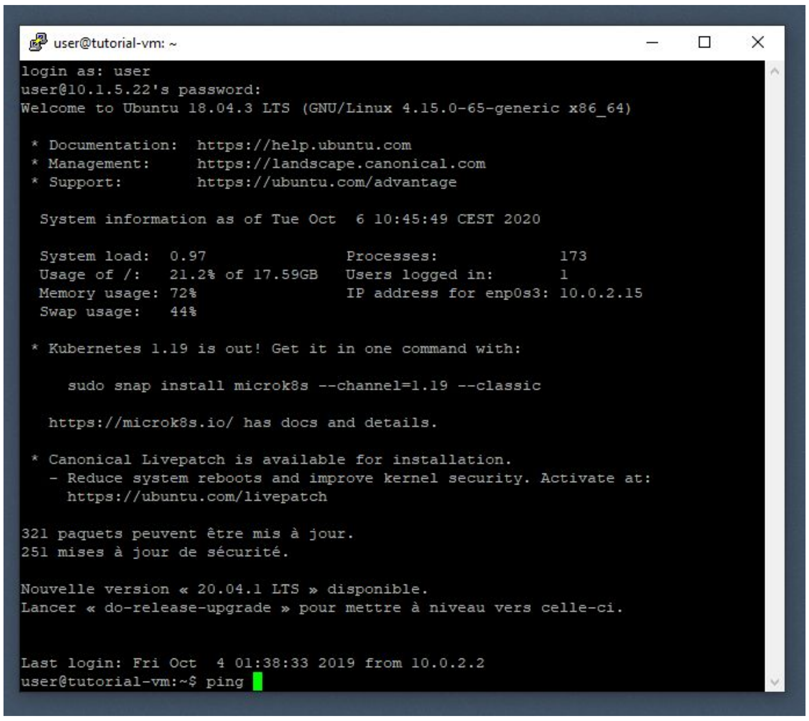
Nom	Protocole	IP hôte	Port hôte	IP invité	Port invité
Rule 1	TCP	10.1.5.22	1234	10.0.2.15	22

To test the Port Forwarding, we use PuTTY as a SSH Putty client on our host machine with the configuration as shown below.



Specify the destination you want to connect to
Host Name (or IP address): 10.1.5.22
Port: 1234
Connection type: Raw Telnet Rlogin [✓ SSH] Serial

As we can see in the figure below, The Port Forwarding allows access to the VM from the host machine. When data packets arrive on port 22 of the host machine, they are directly redirected to the VM (in each system, port 22 is reserved for SSH traffic). The connection to the VM from the outside is now ensured.



```
login as: user
user@10.1.5.22'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 Tue Oct  6 10:45:49 CEST 2020

  System load:  0.97              Processes:             173
  Usage of /:   21.2% of 17.59GB   Users logged in:       1
  Memory usage: 72%               IP address for enp0s3: 10.0.2.15
  Swap usage:   44%

 * Kubernetes 1.19 is out! Get it in one command with:

     sudo snap install microk8s --channel=1.19 --classic

   https://microk8s.io/ has docs and details.

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

321 paquets peuvent être mis à jour.
251 mises à jour de sécurité.

Nouvelle version « 20.04.1 LTS » disponible.
Lancer « do-release-upgrade » pour mettre à niveau vers celle-ci.


Last login: Fri Oct  4 01:38:33 2019 from 10.0.2.2
user@tutorial-vm:~$ ping
```

Fourth part: VM duplication

In this part, we create a new clone of our VM with the same disk file. To do this, we execute the command :

```
VBoxManage clonemedium "C:\Users\m_pages\Downloads\disk.vmdk" "C:\Users\m_pages\Downloads\disk-copy.vmdk"
```

Fifth part: Docker containers provisioning

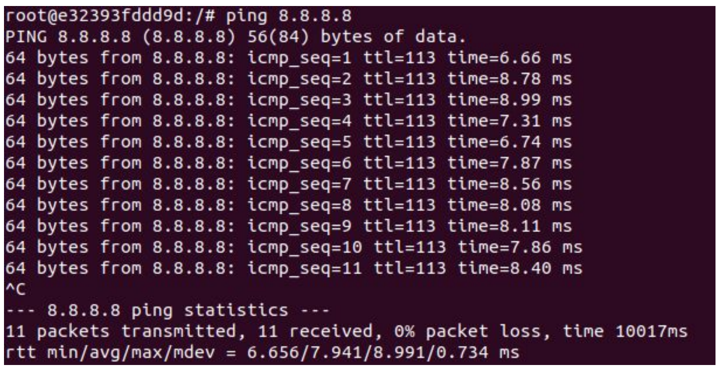
For practical reasons, the target Docker environment is deployed over VirtualBox VM because we have all the admin privileges on these VMs. Indeed, provisioning containers over VMs is technically possible but remains very rare in practice. Containers are usually deployed in bare-metal on physical hosts like it is the case with VMs.

Connectivity *:*

Once the docker is configured using the command lines presented in the topic, we can check the connectivity (through ping) with the newly instantiated Docker.

Docker IP address: 172.17.0.2

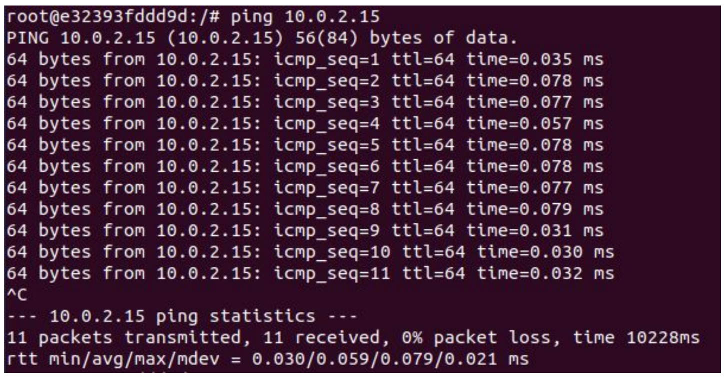
- Ping an Internet resource from Docker



```
root@e32393fddd9d:/# 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=6.66 ms
64 bytes from 8.8.8.8: icmp_seq=2 ttl=113 time=8.78 ms
64 bytes from 8.8.8.8: icmp_seq=3 ttl=113 time=8.99 ms
64 bytes from 8.8.8.8: icmp_seq=4 ttl=113 time=7.31 ms
64 bytes from 8.8.8.8: icmp_seq=5 ttl=113 time=6.74 ms
64 bytes from 8.8.8.8: icmp_seq=6 ttl=113 time=7.87 ms
64 bytes from 8.8.8.8: icmp_seq=7 ttl=113 time=8.56 ms
64 bytes from 8.8.8.8: icmp_seq=8 ttl=113 time=8.08 ms
64 bytes from 8.8.8.8: icmp_seq=9 ttl=113 time=8.11 ms
64 bytes from 8.8.8.8: icmp_seq=10 ttl=113 time=7.86 ms
64 bytes from 8.8.8.8: icmp_seq=11 ttl=113 time=8.40 ms
^C
--- 8.8.8.8 ping statistics ---
11 packets transmitted, 11 received, 0% packet loss, time 10017ms
rtt min/avg/max/mdev = 6.656/7.941/8.991/0.734 ms
```

The connection is ensured from the Docker to an Internet resource.

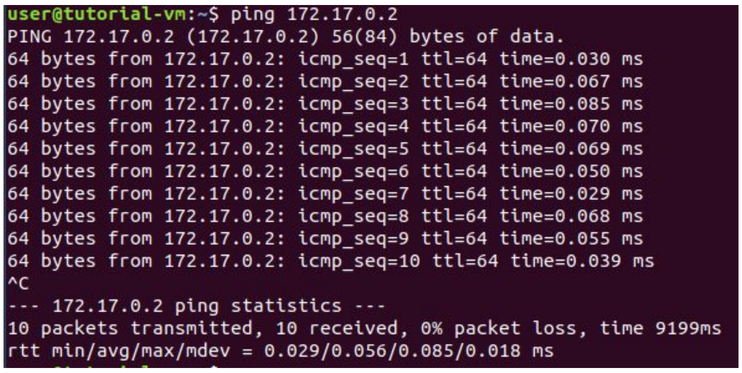
- Ping the VM from Docker



```
root@e32393fddd9d:/# 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=0.035 ms
64 bytes from 10.0.2.15: icmp_seq=2 ttl=64 time=0.078 ms
64 bytes from 10.0.2.15: icmp_seq=3 ttl=64 time=0.077 ms
64 bytes from 10.0.2.15: icmp_seq=4 ttl=64 time=0.057 ms
64 bytes from 10.0.2.15: icmp_seq=5 ttl=64 time=0.078 ms
64 bytes from 10.0.2.15: icmp_seq=6 ttl=64 time=0.078 ms
64 bytes from 10.0.2.15: icmp_seq=7 ttl=64 time=0.077 ms
64 bytes from 10.0.2.15: icmp_seq=8 ttl=64 time=0.079 ms
64 bytes from 10.0.2.15: icmp_seq=9 ttl=64 time=0.031 ms
64 bytes from 10.0.2.15: icmp_seq=10 ttl=64 time=0.030 ms
64 bytes from 10.0.2.15: icmp_seq=11 ttl=64 time=0.032 ms
^C
--- 10.0.2.15 ping statistics ---
11 packets transmitted, 11 received, 0% packet loss, time 10228ms
rtt min/avg/max/mdev = 0.030/0.059/0.079/0.021 ms
```

The connection is ensured from the Docker to the VM.

- 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=0.030 ms
64 bytes from 172.17.0.2: icmp_seq=2 ttl=64 time=0.067 ms
64 bytes from 172.17.0.2: icmp_seq=3 ttl=64 time=0.085 ms
64 bytes from 172.17.0.2: icmp_seq=4 ttl=64 time=0.070 ms
64 bytes from 172.17.0.2: icmp_seq=5 ttl=64 time=0.069 ms
64 bytes from 172.17.0.2: icmp_seq=6 ttl=64 time=0.050 ms
64 bytes from 172.17.0.2: icmp_seq=7 ttl=64 time=0.029 ms
64 bytes from 172.17.0.2: icmp_seq=8 ttl=64 time=0.068 ms
64 bytes from 172.17.0.2: icmp_seq=9 ttl=64 time=0.055 ms
64 bytes from 172.17.0.2: icmp_seq=10 ttl=64 time=0.039 ms
^C
--- 172.17.0.2 ping statistics ---
10 packets transmitted, 10 received, 0% packet loss, time 9199ms
rtt min/avg/max/mdev = 0.029/0.056/0.085/0.018 ms
```

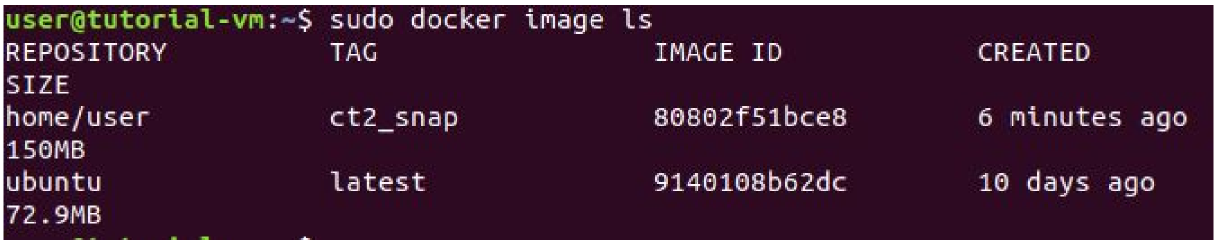
The connection is ensured from the VM to the Docker.

Thanks to these different connectivity tests, we can see that the Docker is correctly configured to the network.

In reality, the Docker is in charge of managing the network. The docker allows to allocate a small part of the VM's network card to different CT..

***Snapshot** :*

After executing a new instance (CT2) of the ubuntu Docker and installing the nano text editor, we make a snapshot of CT2 as shown below.



```
user@tutorial-vm:~$ sudo docker image ls
REPOSITORY          TAG                 IMAGE ID            CREATED
SIZE
home/user           ct2_snap            80802f51bce8        6 minutes ago
150MB
ubuntu              latest              9140108b62dc        10 days ago
72.9MB
```

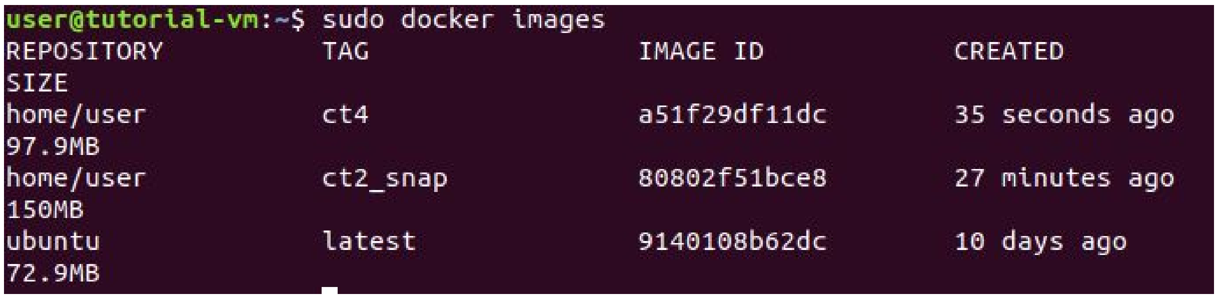
Then, we execute a new instance (CT3) from the snapshot that we previously created from CT2. We use the command line: `$ sudo docker run --name ct3 -it %REPO:%TAG`

After running CT3, we notice that we still have nano installed on CT3. As CT3 was executed from the snapshot of CT2, all parameters of CT2 were preserved, including nano.

Recipes *:*

Docker enables “recipes” sharing to create persistent images (as a second alternative to snapshots). To make a proper recipe, we write a *Dockerfile* following the command lines from the topic and we build the image in the VM.

The recipe CT4 is created, as shown below.

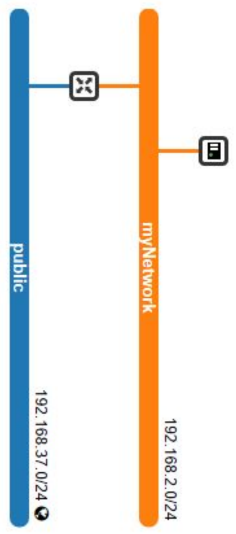


```
user@tutorial-vm:~$ sudo docker images
REPOSITORY          TAG                 IMAGE ID            CREATED
SIZE
home/user           ct4                 a51f29df11dc        35 seconds ago
97.9MB
home/user           ct2_snap            80802f51bce8        27 minutes ago
150MB
ubuntu              latest              9140108b62dc        10 days ago
72.9MB
```

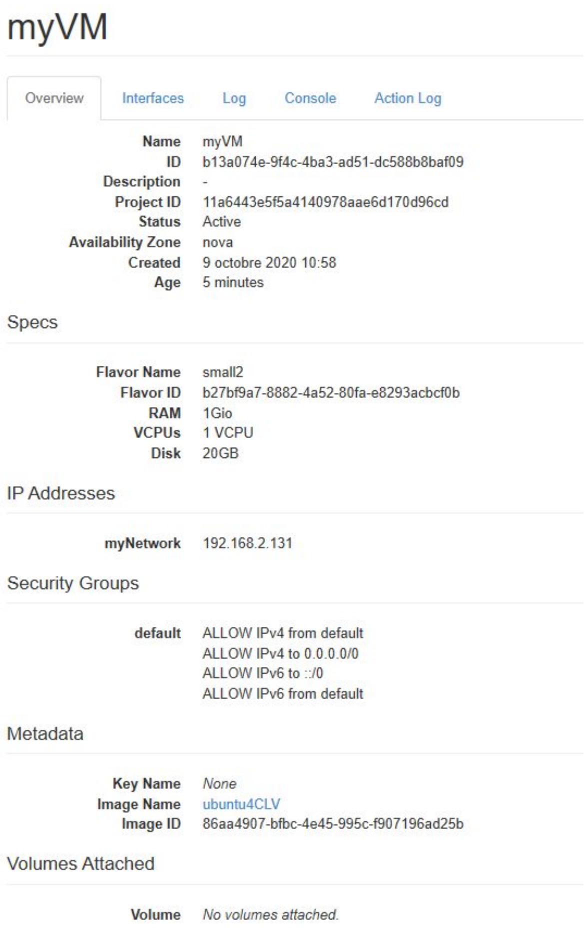
2. Expected work for objectives 6 and 7

First part : CT creation and configuration on OpenStack

In order to create a VM in OpenStack, we have to add a private network on OpenStack because we can't create a VM on a public one. Below is the network topology.



The VM started automatically and all related information is displayed in the dashboard (below).



myVM

Name	myVM
ID	b13a074e-9f4c-4ba3-ad51-dc588b8baf09
Description	-
Project ID	11a6443e5f5a4140978aae6d170d96cd
Status	Active
Availability Zone	nova
Created	9 octobre 2020 10:58
Age	5 minutes

Specs

Flavor Name	small2
Flavor ID	b27bf9a7-8882-4a52-80fa-e8293acbcf0b
RAM	1Gio
VCPUs	1 VCPU
Disk	20GB

IP Addresses

myNetwork	192.168.2.131

Security Groups

default	ALLOW IPv4 from default ALLOW IPv4 to 0.0.0.0/0 ALLOW IPv6 to ::/0 ALLOW IPv6 from default

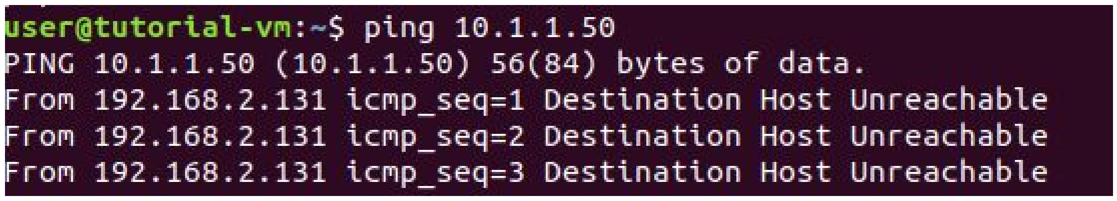
Metadata

Key Name	*None*
Image Name	ubuntu4CLV
Image ID	86aa4907-bfbc-4e45-995c-f907196ad25b

Volumes Attached

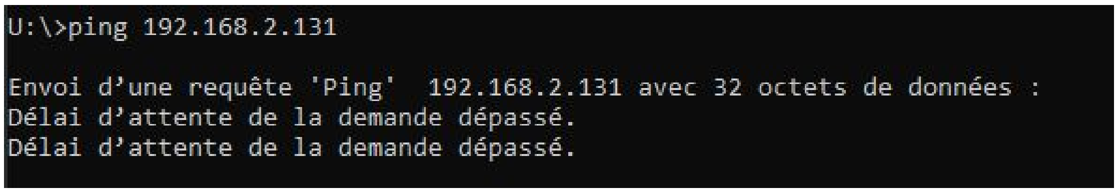
Volume	*No volumes attached.*

We check the connectivity from VM to Host with a Ping:



```
user@tutorial-vm:~$ ping 10.1.1.50
PING 10.1.1.50 (10.1.1.50) 56(84) bytes of data.
From 192.168.2.131 icmp_seq=1 Destination Host Unreachable
From 192.168.2.131 icmp_seq=2 Destination Host Unreachable
From 192.168.2.131 icmp_seq=3 Destination Host Unreachable
```

We check the connectivity from Host to VM with a Ping:



```
U:\>ping 192.168.2.131

Envoi d’une requête 'Ping'  192.168.2.131 avec 32 octets de données :
Délai d’attente de la demande dépassé.
Délai d’attente de la demande dépassé.
```

The Host is unreachable and vice versa, because by default *OpenStack blocks the network traffic on the virtualized private network*.

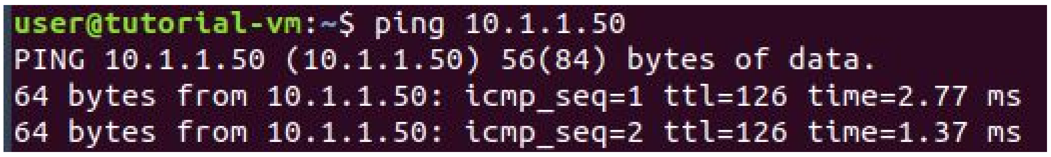
Configuring the security rules in OpenStack:

Because OpenStack blocks the network traffic on the virtualized private network, we have to change the traffic rules. This traffic could be managed using specific security rules that are specific to each project. We authorized the ping (ICMP) and SSH traffic for the rest of the lab through the “Security Groups” section.

Second part: Connectivity test

The VM IP address that was given by the hypervisor is displayed on the dashboard (Instances tab). This address belongs to a private network and is consequently non-routable.

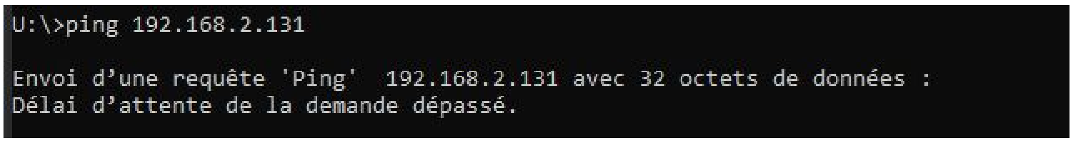
We check the connectivity from the VM to the Host with a Ping:



```
user@tutorial-vm:~$ ping 10.1.1.50
PING 10.1.1.50 (10.1.1.50) 56(84) bytes of data.
64 bytes from 10.1.1.50: icmp_seq=1 ttl=126 time=2.77 ms
64 bytes from 10.1.1.50: icmp_seq=2 ttl=126 time=1.37 ms
```

The ping is working because the network is considered as a sub-network of INSA.

We check the connectivity from the Host to the VM with a Ping:



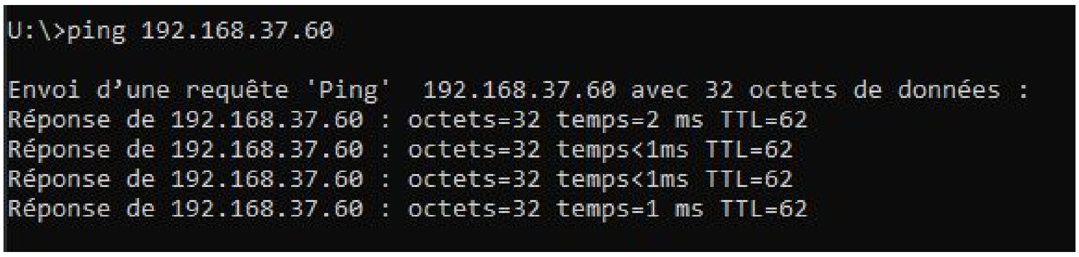
```
U:\>ping 192.168.2.131

Envoi d’une requête 'Ping'  192.168.2.131 avec 32 octets de données :
Délai d’attente de la demande dépassé.
```

The ping is not working, because the VM address is in fact a private network address. It is not reachable. We have to add a floating IP which is public.

We attributed a floating IP which is the following:

Floating IP
New IP address : 192.168.37.60

With the floating IP, the ping from the Host to the VM is now working (below).



```
U:\>ping 192.168.37.60

Envoi d'une requête 'Ping'  192.168.37.60 avec 32 octets de données :
Réponse de 192.168.37.60 : octets=32 temps=2 ms TTL=62
Réponse de 192.168.37.60 : octets=32 temps<1ms TTL=62
Réponse de 192.168.37.60 : octets=32 temps<1ms TTL=62
Réponse de 192.168.37.60 : octets=32 temps=1 ms TTL=62
```

Third part: Snapshot, restore and resize a VM

- First, we tried to resize a running VM from the Instances tab. We observed that we can enlarge the size, but not reduce it because it is a limitation of our OpenStack INSA. Moreover, there is no difference if the VM is on or off. This is a proof of flexibility. However, this flexibility in resizing can cause data loss or process downtime.

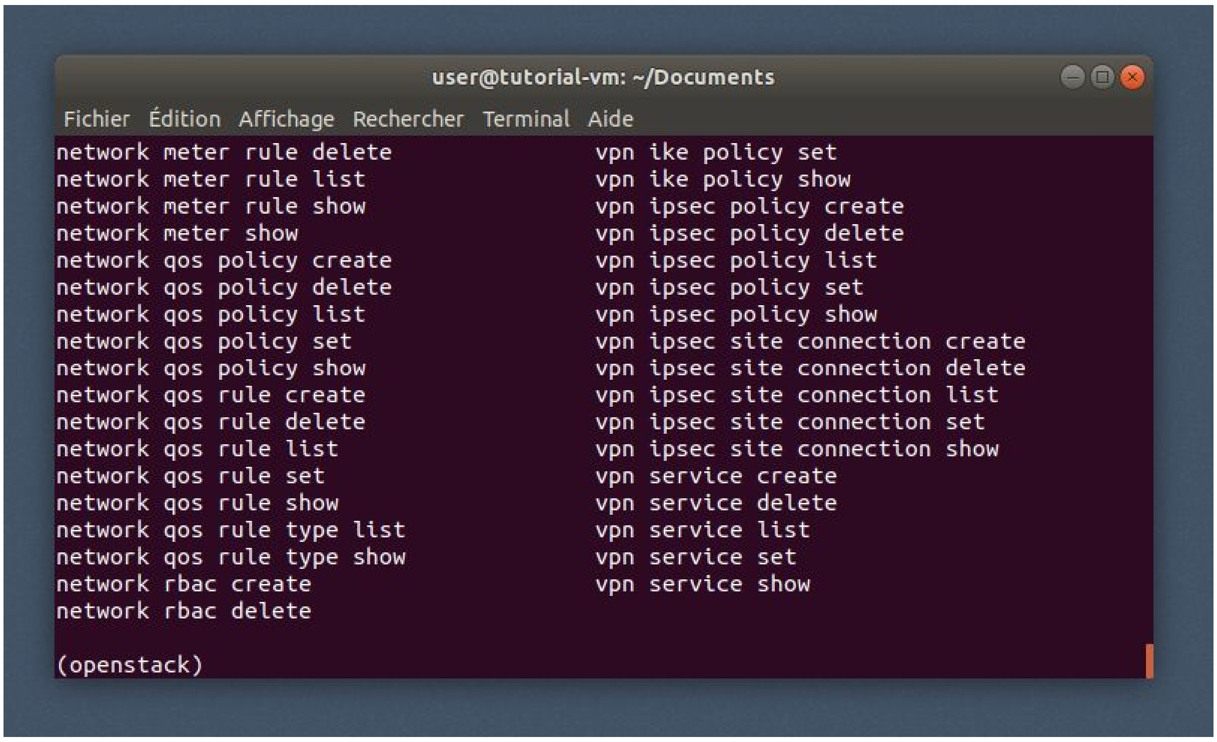
- After making a snapshot of the VM, we can see there are no differences between the included material/software within the original image and the newly created snapshot: images are identical.

- After restoring the VM from the last backup, we saw that the instances are identical.

3. Expected work for objectives 8 and 9

Part one : OpenStack client installation

OpenStack comes with a command line client that enables calling the remote REST operations. In this part, we just configured the client with the appropriate variables. As we can see below, configuration went well.



user@tutorial-vm: ~/Documents
Fichier Édition Affichage Rechercher Terminal Aide

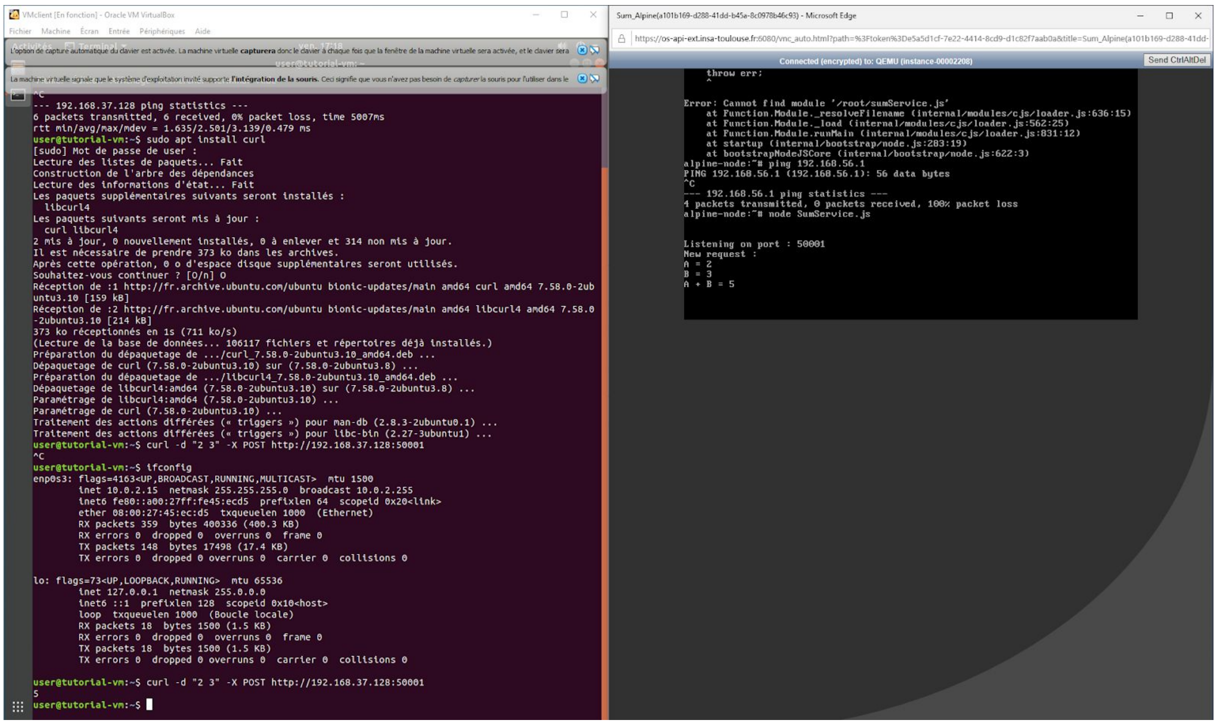
```
network meter rule delete                vpn ike policy set
network meter rule list                  vpn ike policy show
network meter rule show                  vpn ipsec policy create
network meter show                       vpn ipsec policy delete
network qos policy create                vpn ipsec policy list
network qos policy delete                vpn ipsec policy set
network qos policy list                  vpn ipsec policy show
network qos policy set                   vpn ipsec site connection create
network qos policy show                  vpn ipsec site connection delete
network qos rule create                  vpn ipsec site connection list
network qos rule delete                  vpn ipsec site connection set
network qos rule list                    vpn ipsec site connection show
network qos rule set                     vpn service create
network qos rule show                    vpn service delete
network qos rule type list               vpn service list
network qos rule type show               vpn service set
network rbac create                      vpn service show
network rbac delete

(openstack)
```

Part two : Web 2-tier application topology and specification

In this part, we propose to deploy a 2-tier Web application on OpenStack. The application implements a calculator that handles arithmetic operations (i.e. addition, subtraction, multiplication and division) of integer numbers. Each one of these operations is implemented with an NodeJs microservice. In addition, there is a fifth micro-service that implements the application endpoint. Specifically, this service will be listening and receiving the prospective requests when starting the application. Requests are HTTP-based and describe the arithmetic operations that need to be calculated by the application as depicted in Figure 6. The execution result is displayed at both the front-end component and client.

We had a lot of trouble implementing this part, and we had to migrate to Alpine machines at the last minute to get the system up and running.

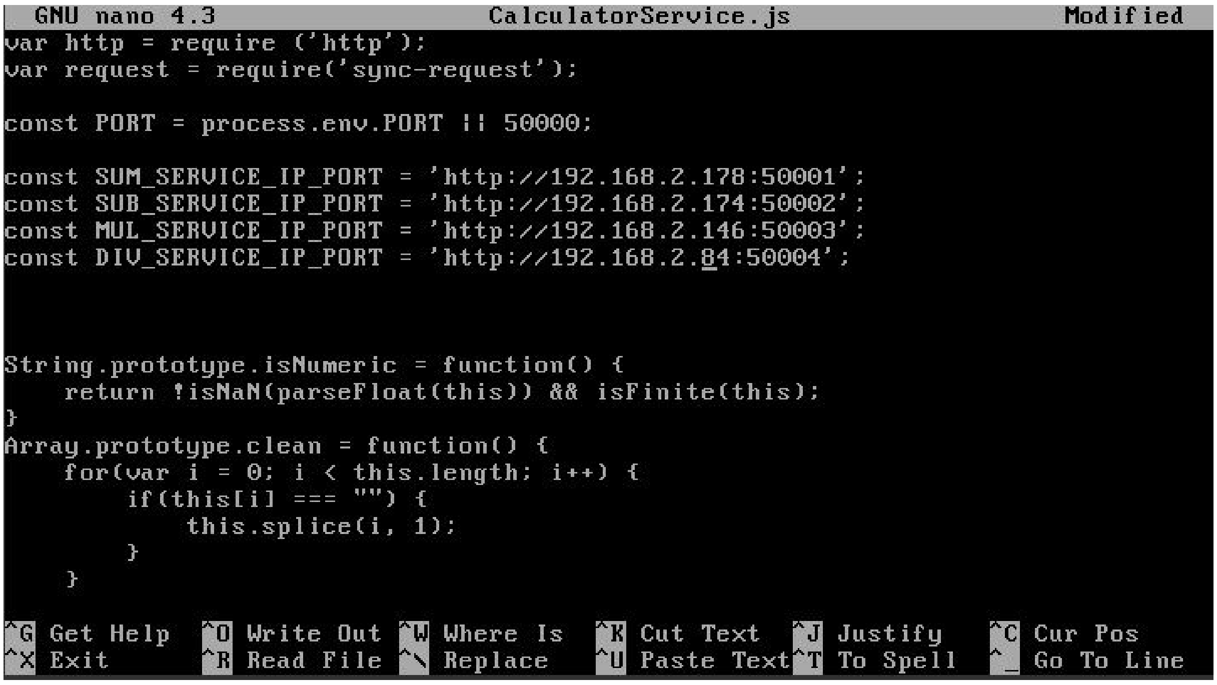
We finally got the system to work properly, as seen below.


VMclient [En fonction] - Oracle VM VirtualBox
Fichier Machine Écran Entrée Périphériques Aide
L'option de capture automatique du clavier est activée. La machine virtuelle **capturera** donc le clavier à chaque fois que la fenêtre de la machine virtuelle sera activée, et le clavier sera
La machine virtuelle signale que le système d'exploitation invité supporte **l'intégration de la souris.** Ceci signifie que vous n'avez pas besoin de *capturer* la souris pour l'utiliser dans le
Sum_Alpine(a101b169-d288-41dd-b45a-8c0978b46c93) - Microsoft Edge
https://os-api-ext.insa-toulouse.fr:6080/vnc_auto.html?path=%3Ftoken%3De5a5d1cf-7e22-4414-8cd9-d1c82f7aab0a&title=Sum_Alpine(a101b169-d288-41dd-
Connected (encrypted) to: QEMU (instance-00002208)
Send CtrlAltDel

```
^C
--- 192.168.37.128 ping statistics ---
6 packets transmitted, 6 received, 0% packet loss, time 5007ms
rtt min/avg/max/mdev = 1.635/2.501/3.139/0.479 ms
user@tutorial-vm:~$ sudo apt install curl
[sudo] Mot de passe de user :
Lecture des listes de paquets... Fait
Construction de l'arbre des dépendances
Lecture des informations d'état... Fait
Les paquets supplémentaires suivants seront installés :
  libcurl4
Les paquets suivants seront mis à jour :
  curl libcurl4
2 mis à jour, 0 nouvellement installés, 0 à enlever et 314 non mis à jour.
Il est nécessaire de prendre 373 ko dans les archives.
Après cette opération, 0 o d'espace disque supplémentaires seront utilisés.
Souhaitez-vous continuer ? [O/n] O
Réception de :1 http://fr.archive.ubuntu.com/ubuntu bionic-updates/main amd64 curl amd64 7.58.0-2ub
untu3.10 [159 kB]
Réception de :2 http://fr.archive.ubuntu.com/ubuntu bionic-updates/main amd64 libcurl4 amd64 7.58.0
-2ubuntu3.10 [214 kB]
373 ko réceptionnés en 1s (711 ko/s)
(Lecture de la base de données... 106117 fichiers et répertoires déjà installés.)
Préparation du dépaquetage de .../curl_7.58.0-2ubuntu3.10_amd64.deb ...
Dépaquetage de curl (7.58.0-2ubuntu3.10) sur (7.58.0-2ubuntu3.8) ...
Préparation du dépaquetage de .../libcurl4_7.58.0-2ubuntu3.10_amd64.deb ...
Dépaquetage de libcurl4:amd64 (7.58.0-2ubuntu3.10) sur (7.58.0-2ubuntu3.8) ...
Paramétrage de libcurl4:amd64 (7.58.0-2ubuntu3.10) ...
Paramétrage de curl (7.58.0-2ubuntu3.10) ...
Traitement des actions différées (« triggers ») pour man-db (2.8.3-2ubuntu0.1) ...
Traitement des actions différées (« triggers ») pour libc-bin (2.27-3ubuntu1) ...
user@tutorial-vm:~$ curl -d "2 3" -X POST http://192.168.37.128:50001
^C
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:fe45:ecd5  prefixlen 64  scopeid 0x20<link>
        ether 08:00:27:45:ec:d5  txqueuelen 1000  (Ethernet)
        RX packets 359  bytes 400336 (400.3 KB)
        RX errors 0  dropped 0  overruns 0  frame 0
        TX packets 148  bytes 17498 (17.4 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 18  bytes 1500 (1.5 KB)
        RX errors 0  dropped 0  overruns 0  frame 0
        TX packets 18  bytes 1500 (1.5 KB)
        TX errors 0  dropped 0 overruns 0  carrier 0  collisions 0

user@tutorial-vm:~$ curl -d "2 3" -X POST http://192.168.37.128:50001
5
user@tutorial-vm:~$ 
```

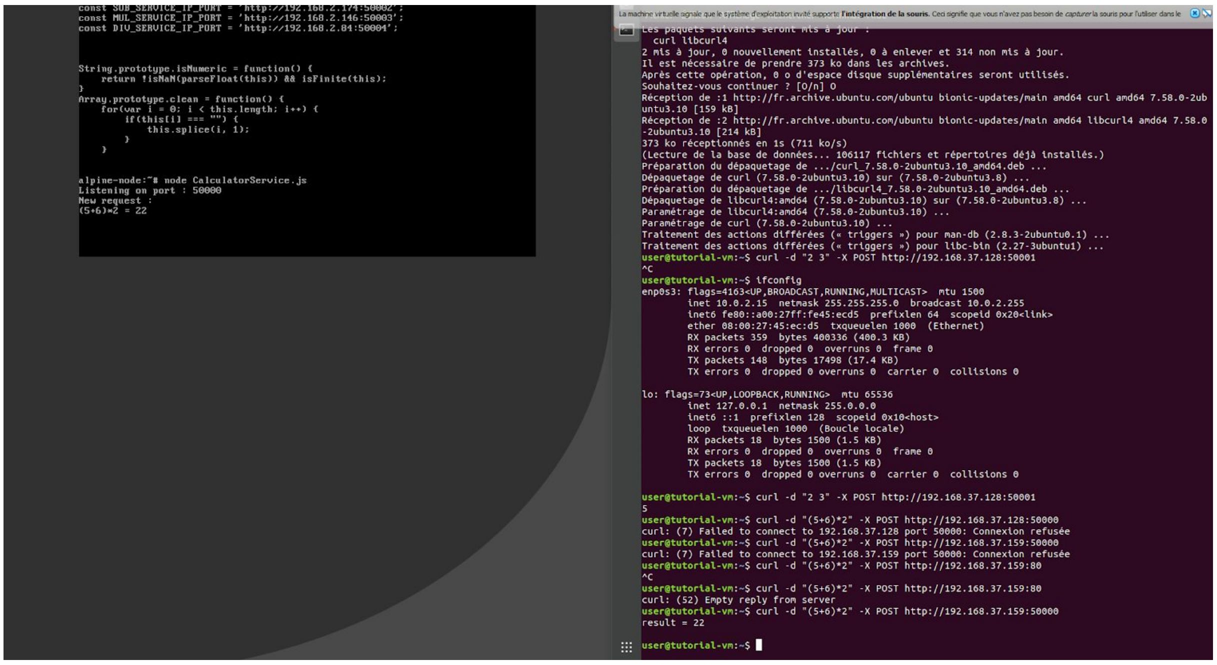
```
    throw err;
    ^

Error: Cannot find module '/root/sumService.js'
    at Function.Module._resolveFilename (internal/modules/cjs/loader.js:636:15)
    at Function.Module._load (internal/modules/cjs/loader.js:562:25)
    at Function.Module.runMain (internal/modules/cjs/loader.js:831:12)
    at startup (internal/bootstrap/node.js:283:19)
    at bootstrapNodeJSCore (internal/bootstrap/node.js:622:3)
alpine-node:~# ping 192.168.56.1
PING 192.168.56.1 (192.168.56.1): 56 data bytes
^C
--- 192.168.56.1 ping statistics ---
4 packets transmitted, 0 packets received, 100% packet loss
alpine-node:~# node SumService.js


Listening on port : 50001
New request :
A = 2
B = 3
A + B = 5
```

One of our problems was that we did not change the port number in the sum service, which we did on the below screenshot.



```
  GNU nano 4.3                  CalculatorService.js                  Modified
var http = require ('http');
var request = require('sync-request');

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

const SUM_SERVICE_IP_PORT = 'http://192.168.2.178:50001';
const SUB_SERVICE_IP_PORT = 'http://192.168.2.174:50002';
const MUL_SERVICE_IP_PORT = 'http://192.168.2.146:50003';
const DIV_SERVICE_IP_PORT = 'http://192.168.2.84:50004';



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);
        }
    }

^G Get Help    ^O Write Out   ^W Where Is    ^K Cut Text    ^J Justify     ^C Cur Pos
^X Exit        ^R Read File   ^\ Replace     ^U Paste Text  ^T To Spell    ^_ Go To Line
```

Finally we got the whole calculator working with all functionalities as below.



```
const SUB_SERVICE_IP_PORT = 'http://192.168.2.174:50002';
const MUL_SERVICE_IP_PORT = 'http://192.168.2.146:50003';
const DIV_SERVICE_IP_PORT = 'http://192.168.2.84:50004';


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);
        }
    }


alpine-node:~# node CalculatorService.js
Listening on port : 50000
New request :
(5+6)*2 = 22
```

La machine virtuelle signale que le système d'exploitation invité supporte **l'intégration de la souris**. Ceci signifie que vous n'avez pas besoin de *capturer* la souris pour l'utiliser dans le

```
Les paquets suivants seront mis à jour :
  curl libcurl4
2 mis à jour, 0 nouvellement installés, 0 à enlever et 314 non mis à jour.
Il est nécessaire de prendre 373 ko dans les archives.
Après cette opération, 0 o d'espace disque supplémentaires seront utilisés.
Souhaitez-vous continuer ? [O/n] O
Réception de :1 http://fr.archive.ubuntu.com/ubuntu bionic-updates/main amd64 curl amd64 7.58.0-2ub
untu3.10 [159 kB]
Réception de :2 http://fr.archive.ubuntu.com/ubuntu bionic-updates/main amd64 libcurl4 amd64 7.58.0
-2ubuntu3.10 [214 kB]
373 ko réceptionnés en 1s (711 ko/s)
(Lecture de la base de données... 106117 fichiers et répertoires déjà installés.)
Préparation du dépaquetage de .../curl_7.58.0-2ubuntu3.10_amd64.deb ...
Dépaquetage de curl (7.58.0-2ubuntu3.10) sur (7.58.0-2ubuntu3.8) ...
Préparation du dépaquetage de .../libcurl4_7.58.0-2ubuntu3.10_amd64.deb ...
Dépaquetage de libcurl4:amd64 (7.58.0-2ubuntu3.10) sur (7.58.0-2ubuntu3.8) ...
Paramétrage de libcurl4:amd64 (7.58.0-2ubuntu3.10) ...
Paramétrage de curl (7.58.0-2ubuntu3.10) ...
Traitement des actions différées (« triggers ») pour man-db (2.8.3-2ubuntu0.1) ...
Traitement des actions différées (« triggers ») pour libc-bin (2.27-3ubuntu1) ...
user@tutorial-vm:~$ curl -d "2 3" -X POST http://192.168.37.128:50001
^C
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:fe45:ecd5  prefixlen 64  scopeid 0x20<link>
        ether 08:00:27:45:ec:d5  txqueuelen 1000  (Ethernet)
        RX packets 359  bytes 400336 (400.3 KB)
        RX errors 0  dropped 0  overruns 0  frame 0
        TX packets 148  bytes 17498 (17.4 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 18  bytes 1500 (1.5 KB)
        RX errors 0  dropped 0  overruns 0  frame 0
        TX packets 18  bytes 1500 (1.5 KB)
        TX errors 0  dropped 0 overruns 0  carrier 0  collisions 0

user@tutorial-vm:~$ curl -d "2 3" -X POST http://192.168.37.128:50001
5
user@tutorial-vm:~$ curl -d "(5+6)*2" -X POST http://192.168.37.128:50000
curl: (7) Failed to connect to 192.168.37.128 port 50000: Connexion refusée
user@tutorial-vm:~$ curl -d "(5+6)*2" -X POST http://192.168.37.159:50000
curl: (7) Failed to connect to 192.168.37.159 port 50000: Connexion refusée
user@tutorial-vm:~$ curl -d "(5+6)*2" -X POST http://192.168.37.159:80
^C
user@tutorial-vm:~$ curl -d "(5+6)*2" -X POST http://192.168.37.159:80
curl: (52) Empty reply from server
user@tutorial-vm:~$ curl -d "(5+6)*2" -X POST http://192.168.37.159:50000
result = 22

user@tutorial-vm:~$ 
```

We ran out of time and we couldn't go any further in this lab. The main difficulty we encountered was technical, with the deployment of the Alpine machine calculator, which was not envisaged at the beginning.

Conclusion

Thanks to the theoretical and practical parts of this lab, we are now able to understand the fundamental differences between the types of hypervisors architecture and between the virtual machines and containers. Moreover, we know how to operate VM with VirtualBox and OpenStack and how to operate Docker containers provisioning, managing the networking connectivity at the same time (NAT mode, Port Forwarding, SSH, etc) .

Finally, the design of a Web 2-tier application at the end of the lab allowed us to understand the notions seen during the whole project. This last exercise was a perfect conclusion to the lab.