UF Infrastructure pour le traitement de données massives

Compte-Rendu de Travaux Pratiques

Concepts et Techniques de Virtualisation (CTV)

2020 - 2021

Redouane Amour
Lilian Aguttes
Zacharie Hellouin

Summary

Objectives

- 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*[1], 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

[1] According to the ETSI terminology.

Theoretical Part :

- 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*[2], 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).

First part: Similarities and differences between the main virtualisation hosts (VM et CT)

We are going to elaborate on two types of virtualisation hosts (i.e. VMs and CTs) in this part. You can see in the table below, our first observations on these two kinds of architecture.

VM	Container
● Real isolation between each OS ● Possibility to choose any OS ● Better security ● Fixed resource allocation	● Forced to use the same OS ● Better performance than VM (CPU & RAM) ● Resource sharing possible ● Dynamic memory allocation

As you can see, each architecture has some advantages/disadvantages. The architecture choice depends on the purpose of your application. If we consider two perspectives, the one of an application developer and the one of an infrastructure administrator, we understand that they value different criterias while choosing their virtualisation host. We are listing and explaining in the table below, which virtualisation host is a better choice for each criteria and each perspective.

Criteria \ POV	Application developer	Infrastructure admin
Virtualization cost (memory size & CPU)	For resource-intensive applications, implementation requires additional software layers. So a developer will prefer containers.	The objective will be to test/develop an infrastructure, without the resources being a real constraint. Both choices are possible
CPU usage, memory, network for given app	Better network addressing for VMs (true @IP), however the reallocation of memory allows better flexibility on performance if using a container. So both choices are possible, depending on your main objective.	If many users are connected, it will be better to have fixed resources already allocated, to avoid management problems. So an administrator will prefer VMs.

[2] According to the ETSI terminology.

Security (access right, rsc sharing)	Less logical security for Containers, however it allows resource sharing between developers. So a developer will prefer containers.	VMs will be more easier to secure, because eachs physical resources are separated. So an administrator will prefer VMs.
Performances (response time)	Resources are shared with containers, so the execution speed will be more useful for a developer. But the response time (time to access the resource) may vary if shared resources are already used per other containers. However, a developer will prefer containers.	With a fixed allocation, VMs allow a constant resources allocation. Thus, even if VMs performances are limited, you obtain a fixed response time with VM, no matter of users numbers connected. So an administrator will prefer VMs.
Tooling for continuous integration support	A container provides native tools for developpement (DevNatives). However, VM's flexibility concerning used OS allows a more versatile test/integration phase. With VM, it is possible to test cross-platform rigorously, using virtual machines with fixed resources	

Second part: 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). In this part, we are going to present and compare mainly two CT technologies, Docker and Linux Containers LXC.

Linux Containers LXC :
Linux LXC is an OS level virtualization process, multiple isolated Linux containers can be run at the same time on a single control host. This CT technology is based on the concept of Linux control groups, the cgroups. LXC provides a virtual environment which offers isolation between each control group, which provides applications with complete isolation of resources, including CPU, memory and I/O accesses.

Docker :
Docker is based on the Linux containers, it used it to add higher level capabilities, to build single application containers. This CT technology builds containers using read only layers of the file system and allows applications to be developed and deployed more quickly, using fewer resources.

Comparison :
Linux LXC is a container technology which provides Linux containers, Docker works differently because it is a single application virtualization process based on containers, like the Linux containers LXC.
LXC behaves like a VM process, it can be treated like an OS, but a Docker cannot have the same behaviour because each Docker container is restricted to a single application. It means that the contenarization level of LXC is at the OS level, for Docker the contenarization level concerns each application which is separate from the others by several containers.
These containers provide a great flexibility to create, deploy and move from one environment to another. Docker containers optimize an application for the cloud, which is not the case of LXC. LXC does not have the same particularities but this CT technology can share and save data in the container or outside.
There are some difficulties with the Docker containers' use. First of all managing a large number of containers at the same time can become a difficult task, moreover containers are isolated but share the same operating system. An attack on the OS can compromise all the containers of the system.
In addition to that with LXC, each container can be used as a sandbox or virtualize a new host. That only depends on the user's need.
The main advantage of Docker is that it is an engine for containers that packs all the applications, along with the dependencies. That means that docker is a portable solution.

Third part: Similarities and differences between Type 1 & Type 2 of hypervisors' architectures

A hypervisor is a hardware virtualization technique that allows multiple OS to run on a single host system at the same time. A hypervisor enables the guest OS to share the hardware of the host, such that each OS appears to have its own processor, memory and other hardware resources. It's also a software (a.k.a Virtual Machine Manager) that allows creating, managing and/or terminating Virtual Machines(VM). There are two types of architectures: Bare-metal (Type 1) and Hosted (Type 2). They offer many of the same characteristics and behaviors like virtualizing the system's available CPU, memory and other resources. Main distinction between these types of architecture comes from how the system installs them.

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

Type 2 Hosted: These hypervisors run on a conventional operating system (OS) just as other computer programs do. A guest operating system runs as a process on the host.

Further in this practical work, we are going to use Virtual Box and OpenStack. These two softwares are hypervisors which belong to different types. Virtual Box is a Type 2 Hypervisor while OpenStack is not a hypervisor. OpenStack is a hypervisor manager that can use different type 1 hypervisors.

Fourth part: Difference between the two main network connection modes for virtualization hosts

(No writing efforts required for this part)

Practical Part :

Tasks related to objectives 4 and 5

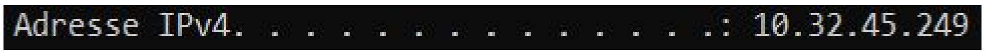
- Objective 4: Operate VirtualBox VMs provisioning, in NAT mode, and make appropriate configurations so that VMs can access and be accessed through the Internet
- Objective 5: Operate Docker containers provisioning, and make appropriate configurations so that containers can access and be accessed through the Internet

First part: Creating and configuring a VM

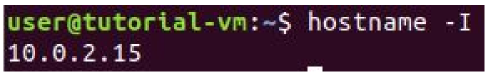
We used VirtualBox to create a VM. With the provided .vmdk, we create a Linux VM. Note that we choose to use a NAT for the Network.

Second part: Testing the VM connectivity

We want to test the connectivity between the VM and the other networks.
Firstly, we compare the host's @IP with the VM's @IP.

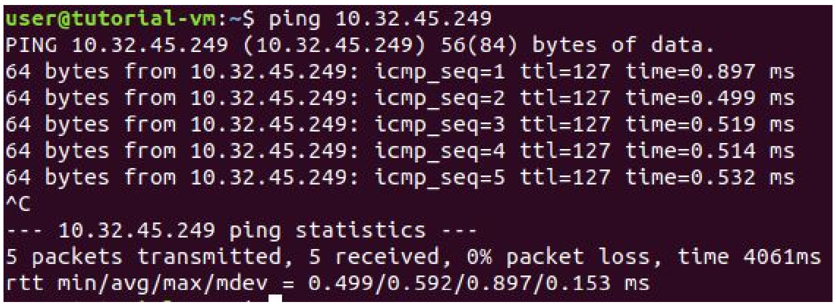
Host (Win) : 

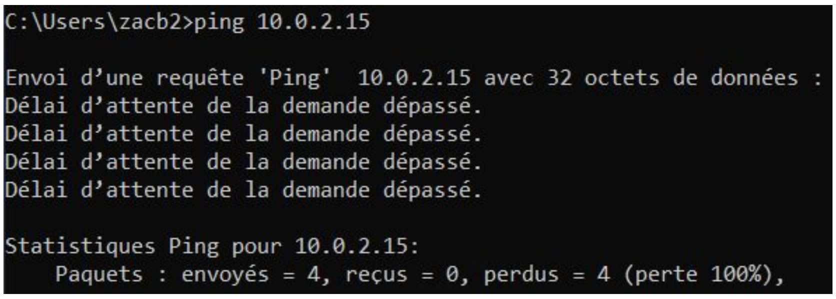
```
Adresse IPv4. . . . . . . . . . . . . . : 10.32.45.249
```

VM (Linux) : 

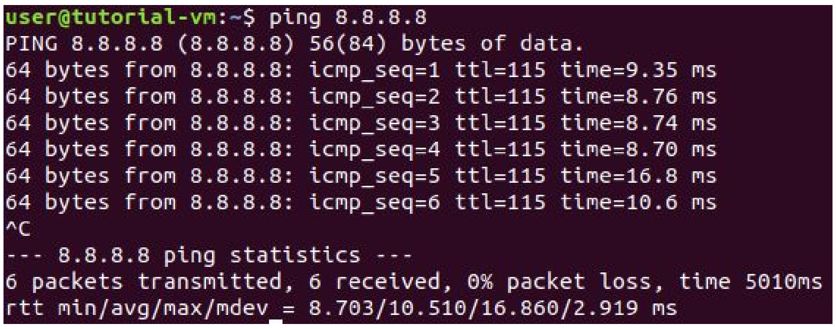
```
user@tutorial-vm:~$ hostname -I
10.0.2.15
```

Connectivity tests :

VM -> Host	 `user@tutorial-vm:~$ ping 10.32.45.249`
	`PING 10.32.45.249 (10.32.45.249) 56(84) bytes of data.`
	`64 bytes from 10.32.45.249: icmp_seq=1 ttl=127 time=0.897 ms`
	`64 bytes from 10.32.45.249: icmp_seq=2 ttl=127 time=0.499 ms`
	`64 bytes from 10.32.45.249: icmp_seq=3 ttl=127 time=0.519 ms`
	`64 bytes from 10.32.45.249: icmp_seq=4 ttl=127 time=0.514 ms`
	`64 bytes from 10.32.45.249: icmp_seq=5 ttl=127 time=0.532 ms`
	`^C`
	`--- 10.32.45.249 ping statistics ---`
	`5 packets transmitted, 5 received, 0% packet loss, time 4061ms`
	`rtt min/avg/max/mdev = 0.499/0.592/0.897/0.153 ms`

Host -> VM	 `C:\Users\zacb2>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.`
	`Délai d'attente de la demande dépassé.`
	`Délai d'attente de la demande dépassé.`
	
	`Statistiques Ping pour 10.0.2.15:`
	`    Paquets : envoyés = 4, reçus = 0, perdus = 4 (perte 100%),`

Neighbour's host -> VM	Doesn't work

VM -> Outside (DNS Google)	

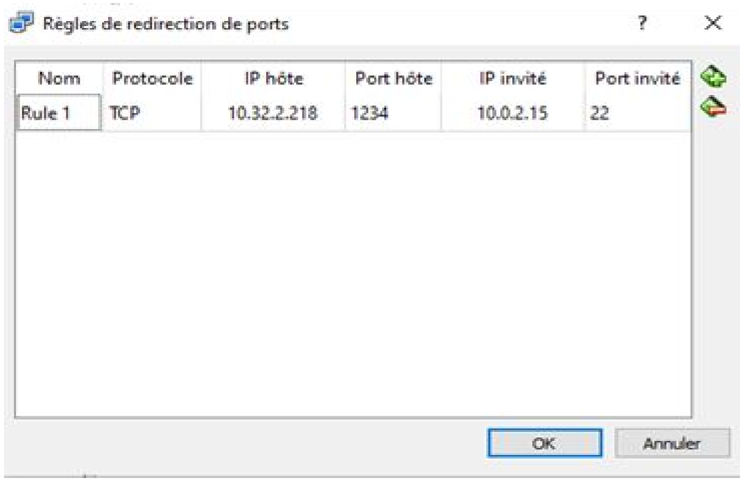
```
user@tutorial-vm:~$ ping 8.8.8.8
PING 8.8.8.8 (8.8.8.8) 56(84) bytes of data.
64 bytes from 8.8.8.8: icmp_seq=1 ttl=115 time=9.35 ms
64 bytes from 8.8.8.8: icmp_seq=2 ttl=115 time=8.76 ms
64 bytes from 8.8.8.8: icmp_seq=3 ttl=115 time=8.74 ms
64 bytes from 8.8.8.8: icmp_seq=4 ttl=115 time=8.70 ms
64 bytes from 8.8.8.8: icmp_seq=5 ttl=115 time=16.8 ms
64 bytes from 8.8.8.8: icmp_seq=6 ttl=115 time=10.6 ms
^C
--- 8.8.8.8 ping statistics ---
6 packets transmitted, 6 received, 0% packet loss, time 5010ms
rtt min/avg/max/mdev = 8.703/10.510/16.860/2.919 ms
```

With this last command, we are trying to ping an exterior server by the name of 8.8.8.8 and we can see that the connection is successful. This connection is made possible by the VirtualBox's Network Adress Translation (NAT) system, from the VM to the exterior.
However, the opposite operation is impossible. In fact, no matter the number of created VMs, they all have the same @IP. It's not a "real" @IP but an integer that simulates/emulates an @IP, that's why it cannot be used to communicate with the VM from the outside. This virtual @IP is not known by the network.
In this manner, most of the VMs with a type 2 supervisor won't be reachable using @IPs.

In the next part, we will fix the identified connectivity issues using Port Forwarding.

Third part: Set up the "missing" connectivity

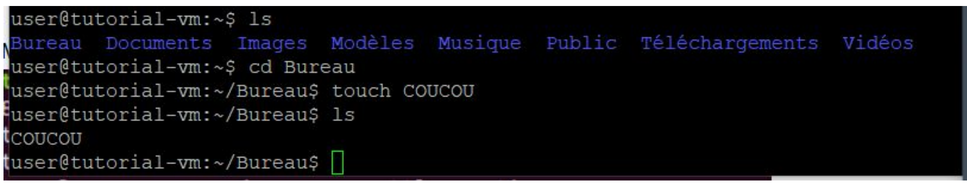
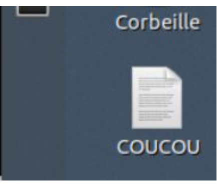
Port forwarding is defined in the VM's properties on VirtualBox.
We define a new connection rule as follows :



Nom	Protocole	IP hôte	Port hôte	IP invité	Port invité
Rule 1	TCP	10.32.2.218	1234	10.0.2.15	22

It's now possible for a specific application from the outside to create a connection to the VM. In order to work, the application must use the port 1234 of the host which will be ***forwarded/translated*** to the VM on port 22.
This rule allows SSH communications (TCP port 22) but not http/https communications (TCP ports 80/443).

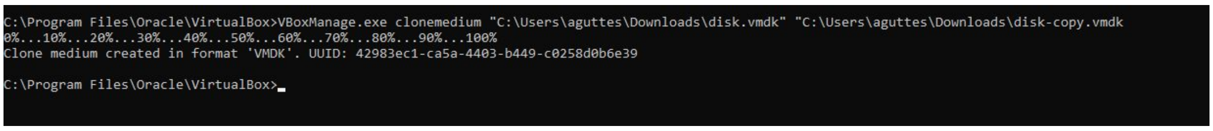
For example,we can now establish an SSH connection from the host to the VM and create a .txt document in the VM’s folder named Desktop :

Host using (Putty is used asa SSH client)	VM
	

```
user@tutorial-vm:~$ ls
Bureau  Documents  Images  Modèles  Musique  Public  Téléchargements  Vidéos
user@tutorial-vm:~$ cd Bureau
user@tutorial-vm:~/Bureau$ touch COUCOU
user@tutorial-vm:~/Bureau$ ls
COUCOU
user@tutorial-vm:~/Bureau$
```

Fourth part: VM duplication

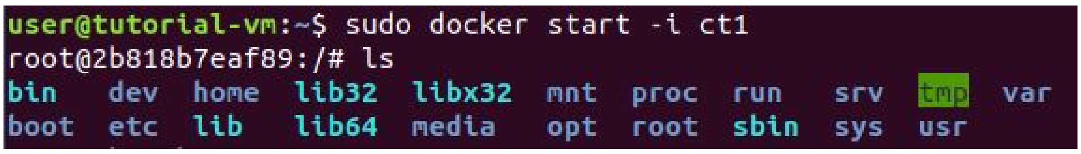
VMs duplication is not as simple as Copy/Paste, because each VM has its own identifier in the hypervisor’s namespace. That’s why a simple copy will create an error in VirtualBox.



```
C:\Program Files\Oracle\VirtualBox>VBoxManage.exe clonemedium "C:\Users\aguttes\Downloads\disk.vmdk" "C:\Users\aguttes\Downloads\disk-copy.vmdk
0%...10%...20%...30%...40%...50%...60%...70%...80%...90%...100%
Clone medium created in format 'VMDK'. UUID: 42983ec1-ca5a-4403-b449-c0258d0b6e39

C:\Program Files\Oracle\VirtualBox>_
```

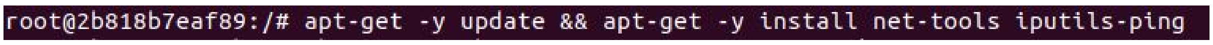
Fifth part: Docker containers provisioning

First, we create the docker containers (CT1, CT2…) inside the Virtual Machine since we have all the admin privileges on these. To do so, we execute the following command :



```
user@tutorial-vm:~$ sudo docker start -i ct1
root@2b818b7eaf89:/# ls
bin   dev  home  lib32  libx32  mnt  proc  run   srv  tmp  var
boot  etc  lib   lib64  media   opt  root  sbin  sys  usr
```

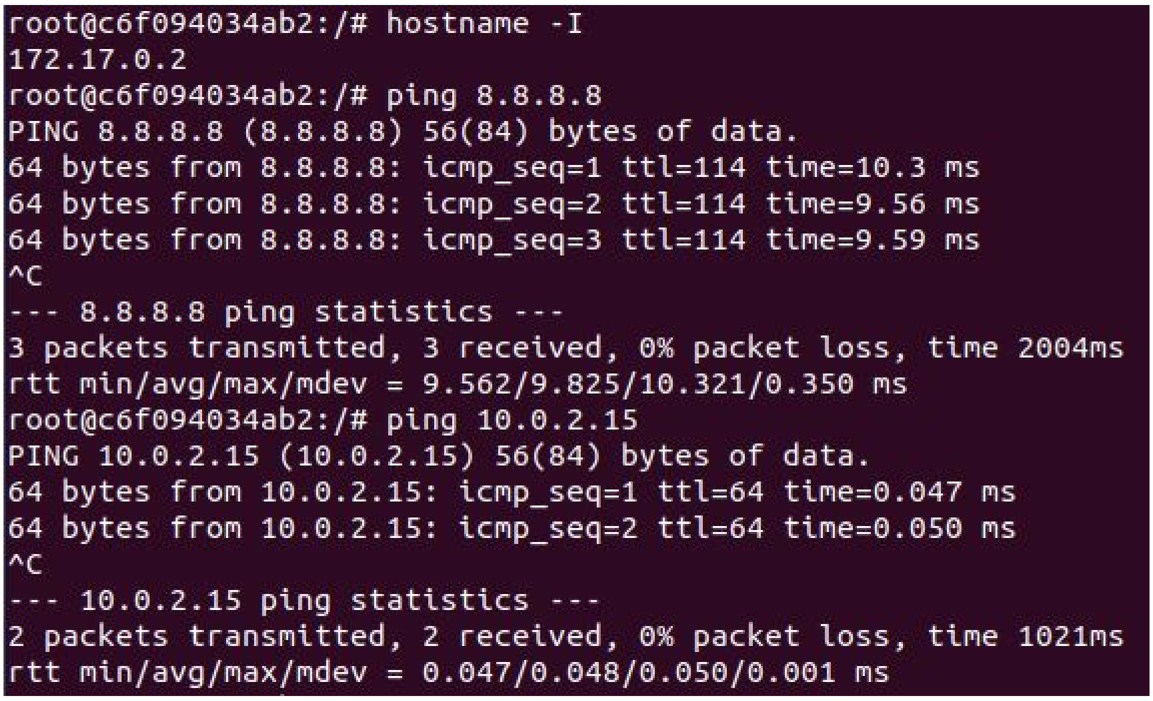
Now that the Docker container is created we can check the connectivity between the docker container and the exterior thanks to the PING command. We first needed to install this command inside the docker container :



```
root@2b818b7eaf89:/# apt-get -y update && apt-get -y install net-tools iputils-ping
```

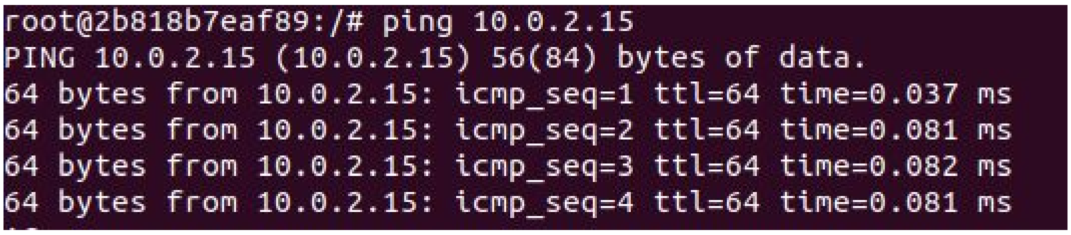
IP adress of the docker container CT1 : 172.17.0.1 (see below).

Docker part : from the docker we try to reach an external server (Google) using the PING command



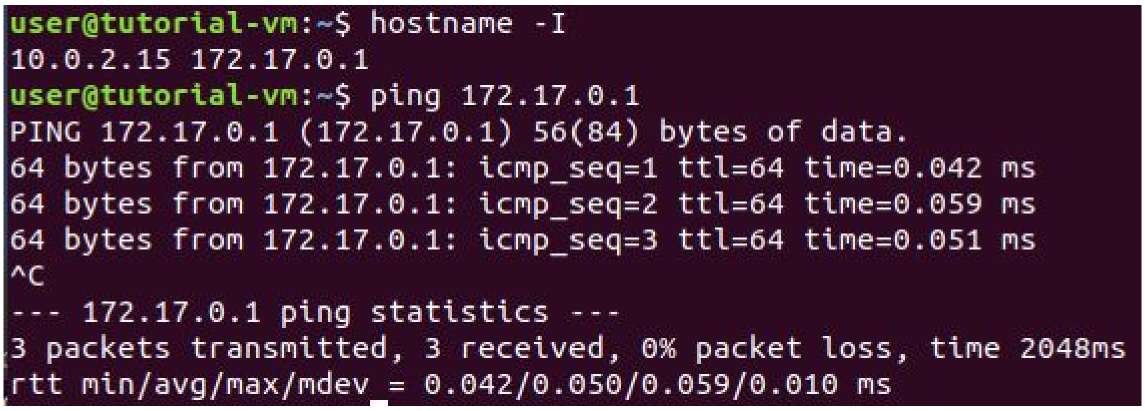
```
root@c6f094034ab2:/# hostname -I
172.17.0.2
root@c6f094034ab2:/# ping 8.8.8.8
PING 8.8.8.8 (8.8.8.8) 56(84) bytes of data.
64 bytes from 8.8.8.8: icmp_seq=1 ttl=114 time=10.3 ms
64 bytes from 8.8.8.8: icmp_seq=2 ttl=114 time=9.56 ms
64 bytes from 8.8.8.8: icmp_seq=3 ttl=114 time=9.59 ms
^C
--- 8.8.8.8 ping statistics ---
3 packets transmitted, 3 received, 0% packet loss, time 2004ms
rtt min/avg/max/mdev = 9.562/9.825/10.321/0.350 ms
root@c6f094034ab2:/# 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=0.047 ms
64 bytes from 10.0.2.15: icmp_seq=2 ttl=64 time=0.050 ms
^C
--- 10.0.2.15 ping statistics ---
2 packets transmitted, 2 received, 0% packet loss, time 1021ms
rtt min/avg/max/mdev = 0.047/0.048/0.050/0.001 ms
```

We can also ping the VM (IP adress : 10.0.2.15) from the docker container :



```
root@2b818b7eaf89:/# 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=0.037 ms
64 bytes from 10.0.2.15: icmp_seq=2 ttl=64 time=0.081 ms
64 bytes from 10.0.2.15: icmp_seq=3 ttl=64 time=0.082 ms
64 bytes from 10.0.2.15: icmp_seq=4 ttl=64 time=0.081 ms
```

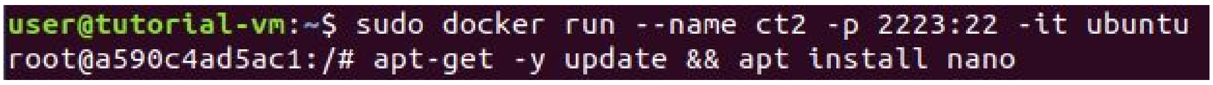
VM part : From the VM (IP address : 10.0.2.15) we ping the docker container (IP address : 172.17.0.1)



```
user@tutorial-vm:~$ hostname -I
10.0.2.15 172.17.0.1
user@tutorial-vm:~$ ping 172.17.0.1
PING 172.17.0.1 (172.17.0.1) 56(84) bytes of data.
64 bytes from 172.17.0.1: icmp_seq=1 ttl=64 time=0.042 ms
64 bytes from 172.17.0.1: icmp_seq=2 ttl=64 time=0.059 ms
64 bytes from 172.17.0.1: icmp_seq=3 ttl=64 time=0.051 ms
^C
--- 172.17.0.1 ping statistics ---
3 packets transmitted, 3 received, 0% packet loss, time 2048ms
rtt min/avg/max/mdev = 0.042/0.050/0.059/0.010 ms
```

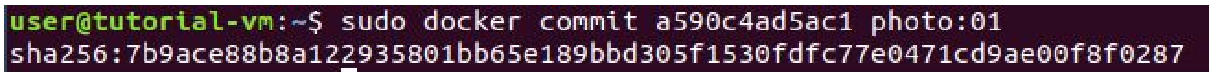
We can see through the former pictures that the connectivity between the docker container inside the VM and the exterior (VM or internet resources) is perfectly established.

Using snapshots of an instance :
In this part, we explore the possibilities offered by docker in terms of instance duplication.
For example, we can create a second container CT2 :



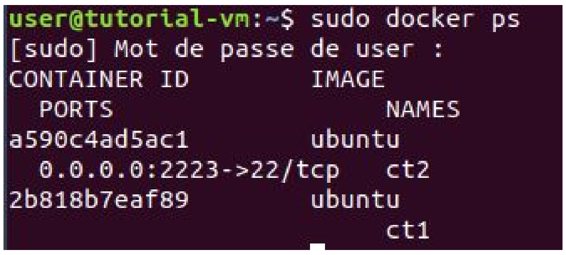
```
user@tutorial-vm:~$ sudo docker run --name ct2 -p 2223:22 -it ubuntu
root@a590c4ad5ac1:/# apt-get -y update && apt install nano
```

After the creation of the second container we can take an image of the CT2 container, in order to have all the parameters, configuration and softwares (like nano) saved on this image. We can do that by using the following command line, after having searched for the containers ID :



```
user@tutorial-vm:~$ sudo docker commit a590c4ad5ac1 photo:01
sha256:7b9ace88b8a122935801bb65e189bbd305f1530fdfc77e0471cd9ae00f8f0287
```

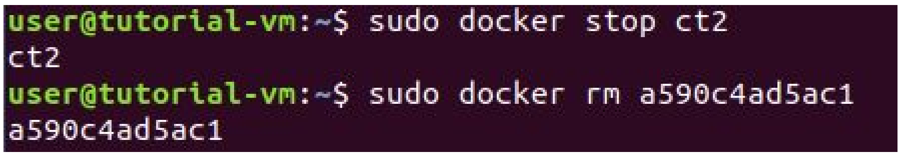
The Docker ID is a590c4ad5ac1, obtained using the following command : sudo docker ps



```
user@tutorial-vm:~$ sudo docker ps
[sudo] Mot de passe de user :
CONTAINER ID        IMAGE
  PORTS                  NAMES
a590c4ad5ac1        ubuntu
  0.0.0.0:2223->22/tcp   ct2
2b818b7eaf89        ubuntu
                         ct1
```

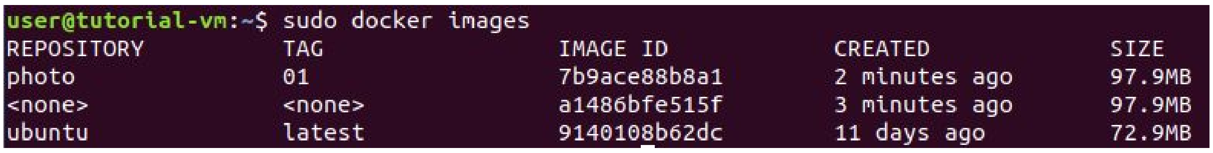
Image Name : photo, using the tag: 01.

After the image is created we can stop and erase the last container, with the following command:



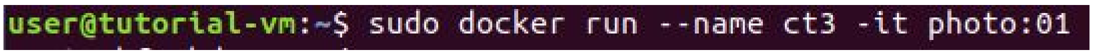
```
user@tutorial-vm:~$ sudo docker stop ct2
ct2
user@tutorial-vm:~$ sudo docker rm a590c4ad5ac1
a590c4ad5ac1
```

We can now list the different snapshots available on the VM and see the one we made earlier (*photo:01*) :



```
user@tutorial-vm:~$ sudo docker images
REPOSITORY          TAG                 IMAGE ID            CREATED             SIZE
photo               01                  7b9ace88b8a1        2 minutes ago       97.9MB
<none>              <none>              a1486bfe515f        3 minutes ago       97.9MB
ubuntu              latest              9140108b62dc        11 days ago         72.9MB
```

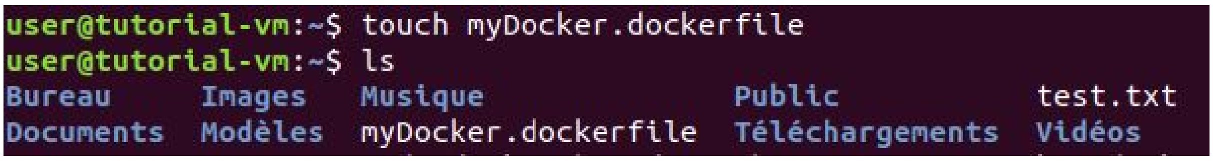
Now we are able to use the previously created snapshot of the CT2 container to create a CT3 container as follows :



```
user@tutorial-vm:~$ sudo docker run --name ct3 -it photo:01
```

If we try to launch nano software, we observe that everything is just the same as it was on the CT2 container, and that is logical because we created the third container using the CT2 snapshot. This snapshot contains all the essential information of CT2.
Every configuration parameter and software initially present on the CT2 container, is also present on the CT3 container.

After checking the specifications of the CT3 container, created using the CT2 snapshot, we can also check how to create a persistent image of a container. To do so, we used the following commands:



```
user@tutorial-vm:~$ touch myDocker.dockerfile
user@tutorial-vm:~$ ls
Bureau     Images   Musique              Public           test.txt
Documents  Modèles  myDocker.dockerfile  Téléchargements  Vidéos
```

We created a .dockerfile file that will contain a sequence of instructions used by docker to create a container with certain parameters and applications.
For example, we can create a *dockerfile* inside the VM, with the following instructions :

```
FROM ubuntu
RUN apt update -y
RUN apt install -y nano
CMD ["/bin/bash"]
```

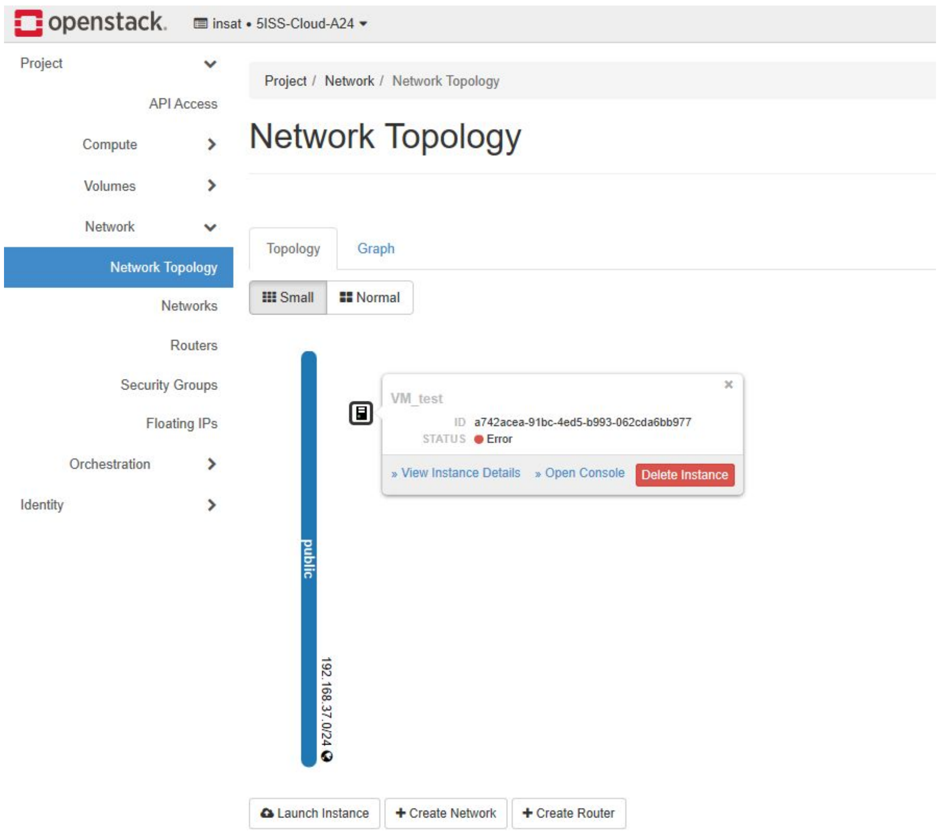
Using this lightweight file, we can create as many containers as we want sharing the same applications and configuration. This dockerfile can be used to generate a container that has nano installed on it.

Tasks related to objectives 6 and 7

- 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).

First part : CT creation and configuration on OpenStack

The purpose of this part is to create a VM on OpenStack and to configure it to be connected on network traffic.



VM_test
ID a742acea-91bc-4ed5-b993-062cda6bb977
STATUS ● Error

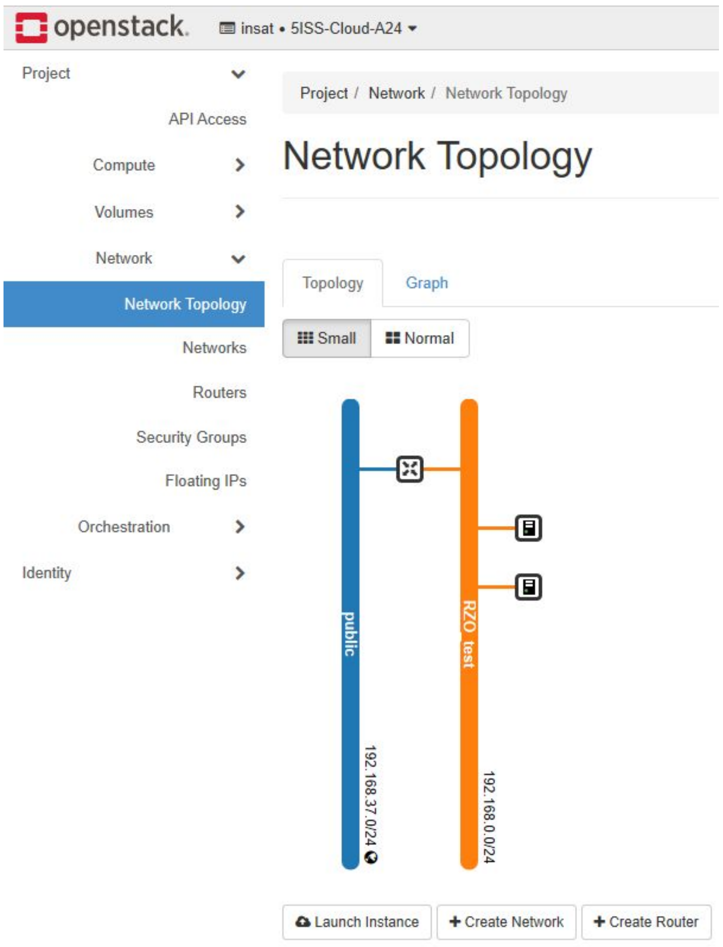
During the VM launching, an error occurred. By default, the VM will try to connect to the public network. Because we don't have the admin rights, openstack denies this kind of connexion. So you have to go through a private network, attached to the public with a router, to connect the VM.

Configuring the security rules in OpenStack

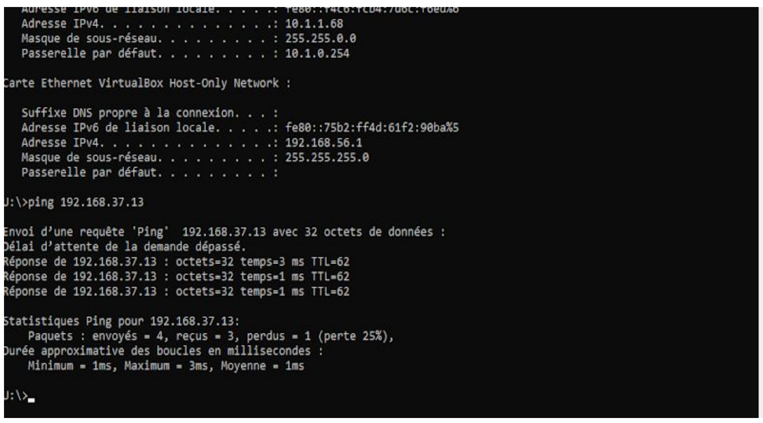
After adding the router, and configuring the security rules in OpenStack (SSH traffic and ICMP for ping tests), we can test the connectivity of our newly created VM.

Second part: Connectivity test

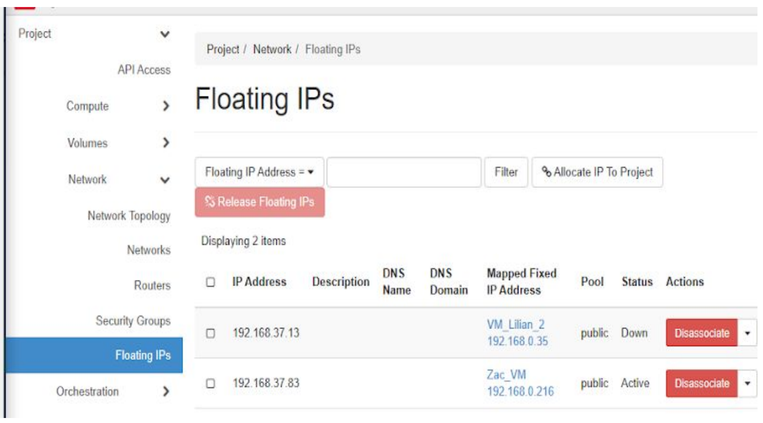
In this part, we are going to try to ping the desktop from the VM, and try again in the opposite direction. The purpose of these operations is to check the connectivity of the VM.



It's impossible to ping the Vm from our physical machine. Indeed, the ip address assigned to our VM is known only on our private network. We must create a floating ip to make this vm accessible from the public domain. (a private IP is non-routable but floating IPs are routable). However it is possible to access the physical machine from our VM because the physical machine ip address is known on the public network (INSA).



```
   Adresse IPv6 de liaison locale. . . . .: fe80::f4c6:fcb4:7d6c:f6ed%6
   Adresse IPv4. . . . . . . . . . . . . .: 10.1.1.68
   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::75b2:ff4d:61f2:90ba%5
   Adresse IPv4. . . . . . . . . . . . . .: 192.168.56.1
   Masque de sous-réseau. . . . . . . . . : 255.255.255.0
   Passerelle par défaut. . . . . . . . . :

U:\>ping 192.168.37.13

Envoi d'une requête 'Ping'  192.168.37.13 avec 32 octets de données :
Délai d'attente de la demande dépassé.
Réponse de 192.168.37.13 : octets=32 temps=3 ms TTL=62
Réponse de 192.168.37.13 : octets=32 temps=1 ms TTL=62
Réponse de 192.168.37.13 : octets=32 temps=1 ms TTL=62

Statistiques Ping pour 192.168.37.13:
    Paquets : envoyés = 4, reçus = 3, perdus = 1 (perte 25%),
Durée approximative des boucles en millisecondes :
    Minimum = 1ms, Maximum = 3ms, Moyenne = 1ms

U:\>_
```



Floating IPs

Displaying 2 items

IP Address	Description	DNS Name	DNS Domain	Mapped Fixed IP Address	Pool	Status	Actions
192.168.37.13				VM_Lilian_2 192.168.0.35	public	Down	Disassociate
192.168.37.83				Zac_VM 192.168.0.216	public	Active	Disassociate

When the configuration of the floating ip is done, we are able to ping our VM, as you can see above.

Third part: Snapshot, restore and resize a VM

We are going to apply some operations on our VM in this part. Before starting, we make a snapshot of the VM. A snapshot is useful when you want to duplicate a VM, within keeping all softwares/materials already installed in it.

When we try to decrease the size of the VM, whether it is running or not, the system returns an error (see below). Indeed, it is a critical operation for it. Unfortunately, it may delete data to format his space to the one requested. In the other direction (enlargement of the available space), the operation does not involve any risky operation and works properly.

Sometimes, these operations work but we cannot conclude that the implementation is safe on OpenStack.

```
Error: Unexpected API Error. Please report this at http://bugs.launchpad.net/nova/ and
attach the Nova API log if possible. <class 'nova.exception.FlavorDiskSmallerThanMinDisk'>
(HTTP 500) (Request-ID: req-01409e51-c051-4afc-b5c4-803ebbc34665)
```

At the end, we restore our VM thanks to the snapshot made at the beginning.

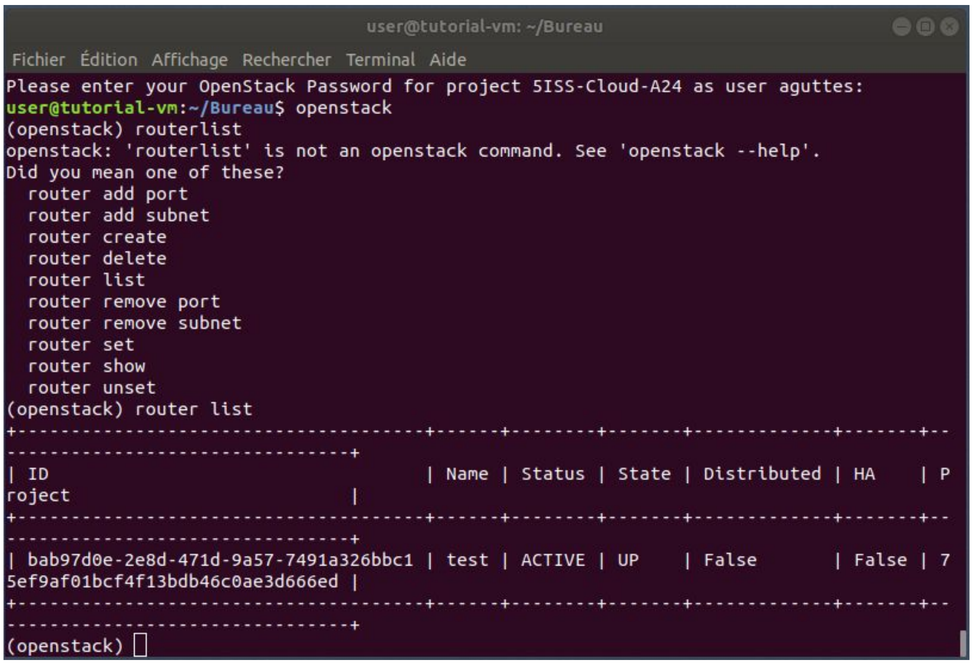
Expected work for objectives 8 and 9

- 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.

Part one : OpenStack client installation

It may be useful to create an Openstack client to interface with openstack directly on our terminal. Especially if you want to use scripts in order to automate VM administration operations. This is what we are exploring in this part.

We have to retrieve the configuration file from the openstack page. This file. sh, contains all the required information (Openstack address, port, project…) for setting the OpenStackClient that we have downloaded on our local VM before. Now the client is set, you can locally execute it from a terminal, and have access to the service. However, this client does not allow direct access to VMs.

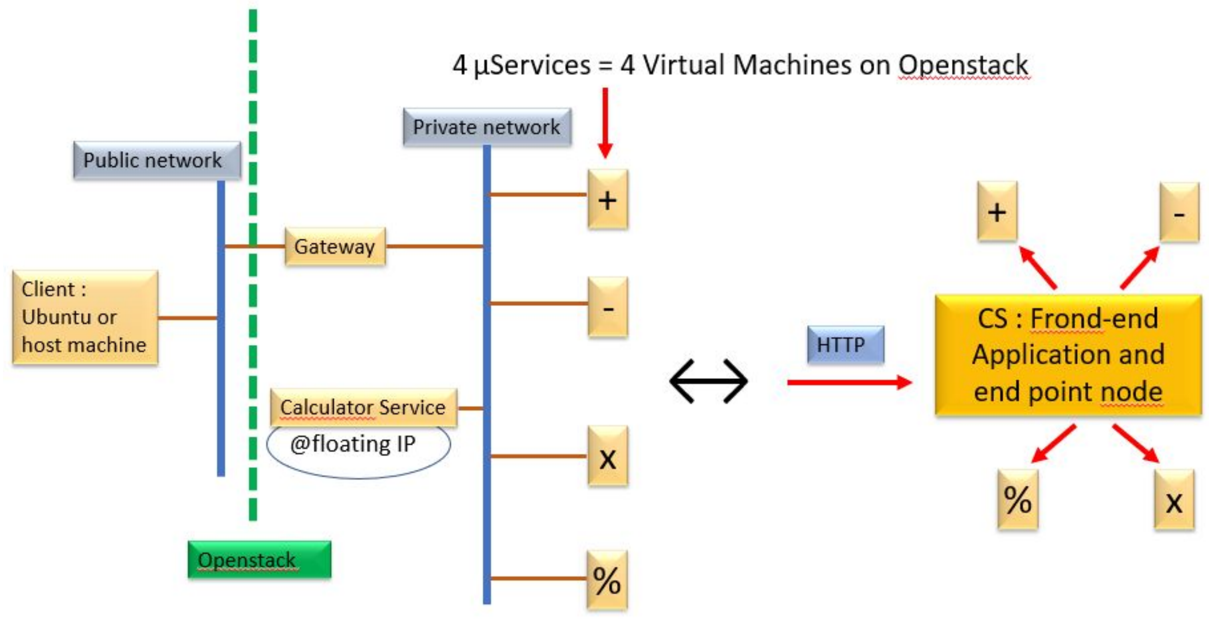

user@tutorial-vm: ~/Bureau
Fichier Édition Affichage Rechercher Terminal Aide

```
Please enter your OpenStack Password for project 5ISS-Cloud-A24 as user aguttes:
user@tutorial-vm:~/Bureau$ openstack
(openstack) routerlist
openstack: 'routerlist' is not an openstack command. See 'openstack --help'.
Did you mean one of these?
  router add port
  router add subnet
  router create
  router delete
  router list
  router remove port
  router remove subnet
  router set
  router show
  router unset
(openstack) router list
+--------------------------------------+------+--------+-------+-------------+-------+--
--------------------------------+
| ID                                   | Name | Status | State | Distributed | HA    | P
roject                          |
+--------------------------------------+------+--------+-------+-------------+-------+--
--------------------------------+
| bab97d0e-2e8d-471d-9a57-7491a326bbc1 | test | ACTIVE | UP    | False       | False | 7
5ef9af01bcf4f13bdb46c0ae3d666ed |
+--------------------------------------+------+--------+-------+-------------+-------+--
--------------------------------+
(openstack) 
```

Once connected to our openstack account, we find the previous created elements. Above, we are showing the router named “test” that allows the connection between our private and public network.

Part two : Web 2-tier application topology and specification

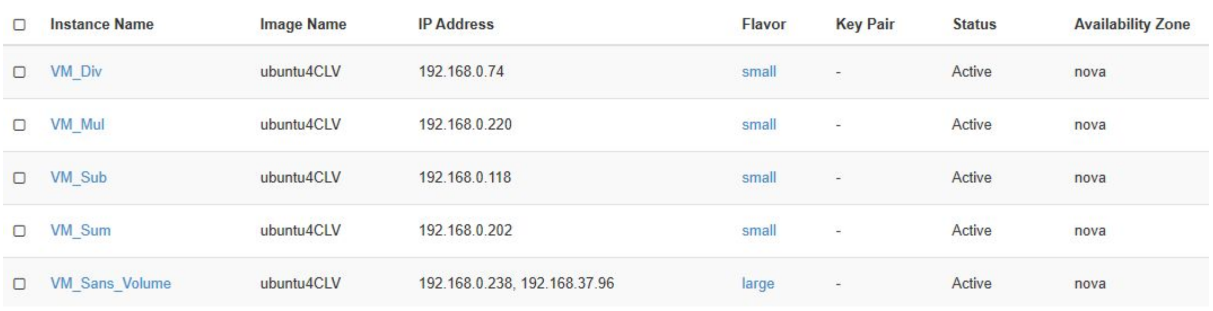
The goal of this part is to use several micro-services initially, and subsequently link them to a "management" service, the front end. Once the runtimes needed to run the services are downloaded, we launch the so-called "basic" services. We are going to create 5 virtual machines on openstack : a VM for the management service called the "calculator service" (see below) which can be identified as both a front-end application and an end-point-node, that listens and receives HTTP requests from the public network. And 4 other virtual machines called microservices representing the 4 elementary operations.
Here is a little scheme to understand our purpose :


4 µServices = 4 Virtual Machines on Openstack
Private network
Public network
Gateway
Client : Ubuntu or host machine
Calculator Service
@floating IP
Openstack
+
-
x
%
HTTP
CS : Frond-end Application and end point node
+
-
%
x

To emulate those services we needed to install : NodeJs, Npm and cURL. For each microservices we completed the following steps :

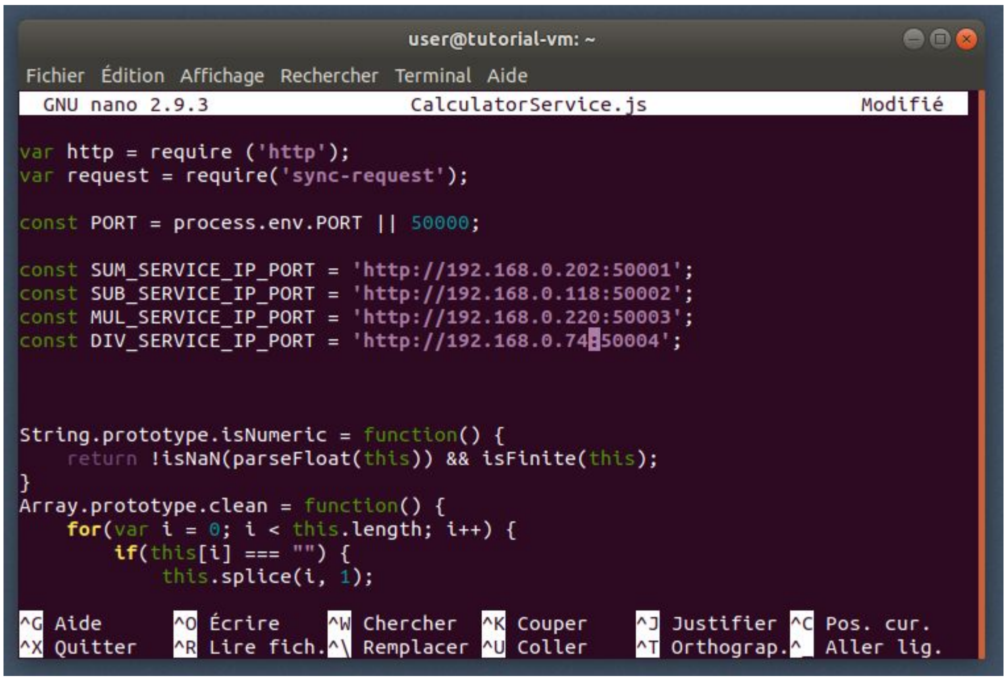
- configuration of the french keyboard
- use of the command "sudo apt install nodejs"
- use of the command "sudo apt install npm"
- use of the command "sudo apt install curl"
- And finally use of the command "wget http://homepages.laas.fr/smedjiah/tmp/operatorService.js where operator designates the 4 elementary operations.

We tried to use the duplication of VMs based on snapshots but it didn't work as intended.



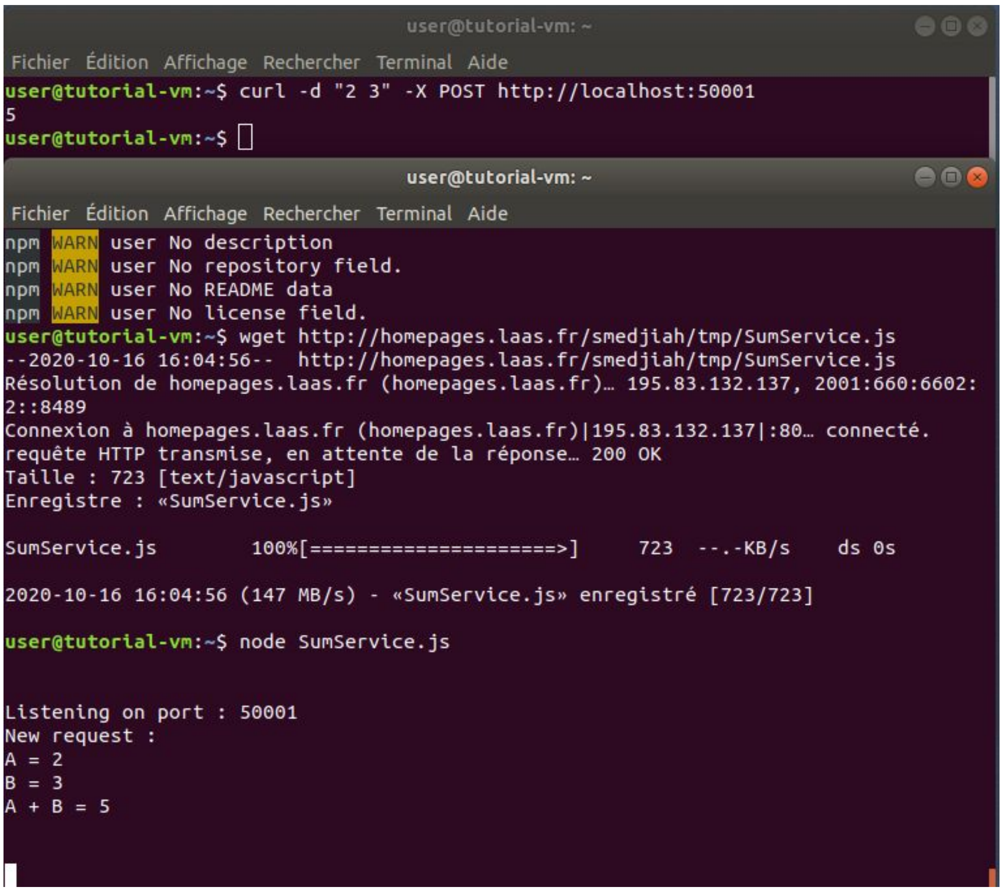
☐	Instance Name	Image Name	IP Address	Flavor	Key Pair	Status	Availability Zone
☐	VM_Div	ubuntu4CLV	192.168.0.74	small	-	Active	nova
☐	VM_Mul	ubuntu4CLV	192.168.0.220	small	-	Active	nova
☐	VM_Sub	ubuntu4CLV	192.168.0.118	small	-	Active	nova
☐	VM_Sum	ubuntu4CLV	192.168.0.202	small	-	Active	nova
☐	VM_Sans_Volume	ubuntu4CLV	192.168.0.238, 192.168.37.96	large	-	Active	nova

In order to call out the previously defined services, the client needs to reach the Calculator service through the network, which will then call the web services installed on the 4 virtual machines. We thus had to configure each IP address for each VM through the CS script and give the calculator service a floating address with a reachable port ranging from 50 000 to 50 050.



```
  GNU nano 2.9.3                 CalculatorService.js                  Modifié

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

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

const SUM_SERVICE_IP_PORT = 'http://192.168.0.202:50001';
const SUB_SERVICE_IP_PORT = 'http://192.168.0.118:50002';
const MUL_SERVICE_IP_PORT = 'http://192.168.0.220:50003';
const DIV_SERVICE_IP_PORT = 'http://192.168.0.74:50004';



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 Aide      ^O Écrire     ^W Chercher  ^K Couper    ^J Justifier ^C Pos. cur.
^X Quitter   ^R Lire fich.^\ Remplacer ^U Coller    ^T Orthograp.^_ Aller lig.
```

Use example : In the top window, we send a request dedicated to the service located on port 50001 of our virtual machine. This port corresponds to the sum service. Thus, we see on the second window, where we started the execution of this service, that it receives the request and returns the response, namely 5.


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

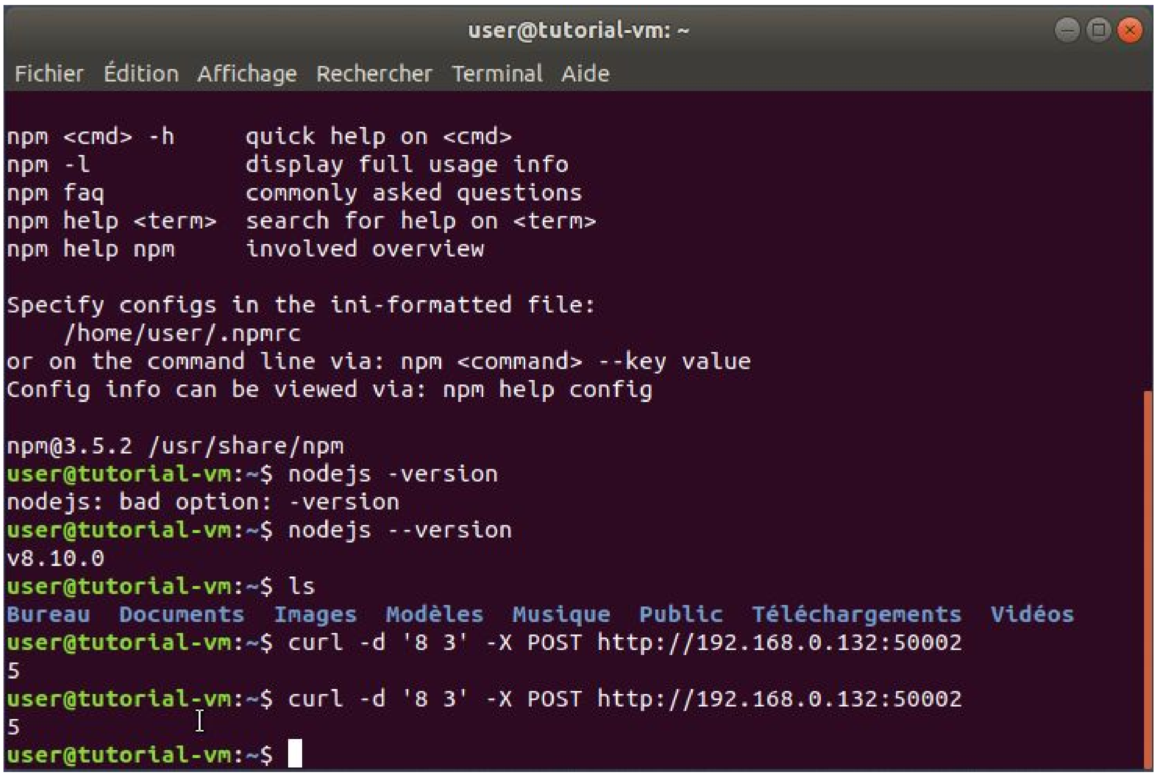
```
user@tutorial-vm:~$ curl -d "2 3" -X POST http://localhost:50001
5
user@tutorial-vm:~$ 
```

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

```
npm WARN user No description
npm WARN user No repository field.
npm WARN user No README data
npm WARN user No license field.
user@tutorial-vm:~$ wget http://homepages.laas.fr/smedjiah/tmp/SumService.js
--2020-10-16 16:04:56--  http://homepages.laas.fr/smedjiah/tmp/SumService.js
Résolution de homepages.laas.fr (homepages.laas.fr)… 195.83.132.137, 2001:660:6602:
2::8489
Connexion à homepages.laas.fr (homepages.laas.fr)|195.83.132.137|:80… connecté.
requête HTTP transmise, en attente de la réponse… 200 OK
Taille : 723 [text/javascript]
Enregistre : «SumService.js»

SumService.js        100%[=====================>]     723  --.-KB/s    ds 0s

2020-10-16 16:04:56 (147 MB/s) - «SumService.js» enregistré [723/723]

user@tutorial-vm:~$ node SumService.js


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

Part three: Deploy the Calculator application on OpenStack

After creating 5 VMs on OpenStack from the same snapshot, we installed a dedicated service on each VM. In the window below, we call the SUB service from another VM. The service response is consistent with the result of the requested subtraction.



```
npm <cmd> -h     quick help on <cmd>
npm -l           display full usage info
npm faq          commonly asked questions
npm help <term>  search for help on <term>
npm help npm     involved overview

Specify configs in the ini-formatted file:
    /home/user/.npmrc
or on the command line via: npm <command> --key value
Config info can be viewed via: npm help config

npm@3.5.2 /usr/share/npm
user@tutorial-vm:~$ nodejs -version
nodejs: bad option: -version
user@tutorial-vm:~$ nodejs --version
v8.10.0
user@tutorial-vm:~$ ls
Bureau  Documents  Images  Modèles  Musique  Public  Téléchargements  Vidéos
user@tutorial-vm:~$ curl -d '8 3' -X POST http://192.168.0.132:50002
5
user@tutorial-vm:~$ curl -d '8 3' -X POST http://192.168.0.132:50002
5
user@tutorial-vm:~$ 
```

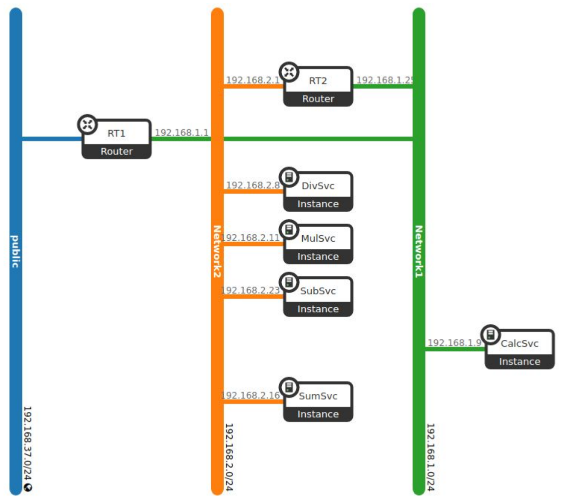
Unfortunately, we didn’t succeed to deploy the calculator application on Openstack. Indeed, when we tried to launch the calculator service, some errors occurred.

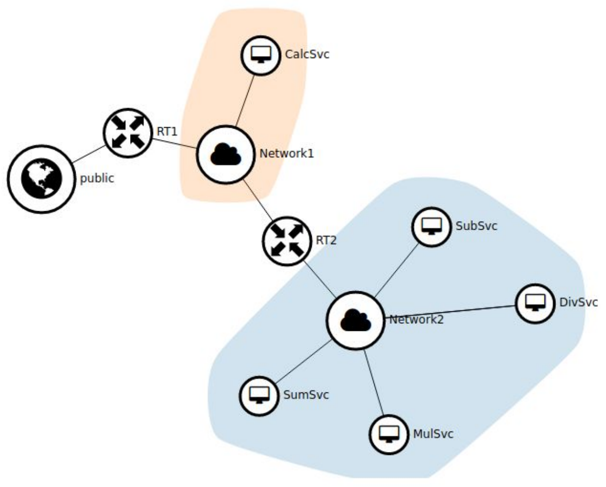
Expected work for objectives 10 and 11

- Objective 10 : Make up a specific network topology on OpenStack
- Objective 11: Configure and deploy this network topology

Part one: The client requirements and target network topology

Using the same services as before, we are asked to deploy the following architecture. It differs mainly in terms of network architecture.





Part two: Deployment of the target topology

For the deployment, we proceed just as in the previous part but with an additional router.

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