

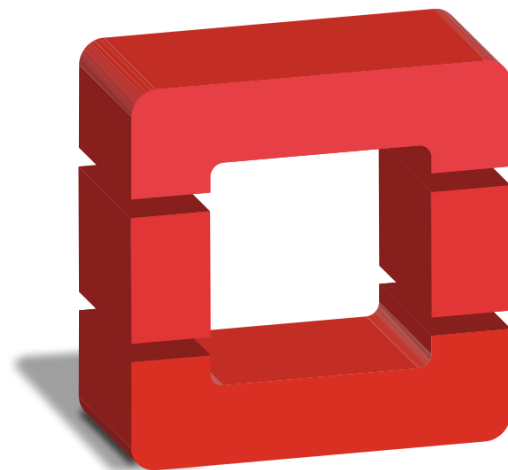
CLOUD COMPUTING : ADAPTABILITY AND AUTONOMIC MANAGEMENT

LAB 1 : INTRODUCTION TO CLOUD HYPERVISORS

24 novembre 2021

Teo GENEAU
Yosra ZEYRI
INSA Toulouse

SISS Groupe B1
Sami YANGUI



openstack®
CLOUD SOFTWARE

TABLE OF CONTENTS

| | |
|--|----|
| TABLE OF CONTENTS | 2 |
| INTRODUCTION | 4 |
| THEORETICAL PART | 5 |
| 1 - Similarities and differences between the main virtualisation hosts (VM et CT) | 5 |
| Similarities and differences between the existing CT types..... | 8 |
| 3. Similarities and differences between Type 1 & Type 2 of hypervisors' architectures..... | 8 |
| PRACTICAL PART | 10 |
| 2. Tasks related to objectives 4 and 5..... | 10 |
| 2.1 First part: Creating and configuring a VM | 10 |
| 2.2 Testing the VM connectivity..... | 10 |
| 2.3 Setting up the "missing" connectivity..... | 11 |
| 2.4 Duplicating the VM..... | 12 |
| 2.5 Docker containers provisioning | 13 |
| 2.5.1 Multiple containers..... | 14 |
| 3. Tasks related to objectives 6 and 7..... | 16 |
| 3.2 Connectivity test | 16 |
| 3.3 Snapshot, restore and resize a VM..... | 18 |
| 4. Tasks related to objectives 8 and 9..... | 19 |
| 4.1 OpenStack client installation | 19 |
| 4.2. Web 2-tier application topology and specification | 19 |
| 4.3 Deploy the Calculator application on OpenStack | 20 |
| 5. Tasks related to objectives 10 and 11..... | 22 |
| 5.1 Configuration of the topology and execution of the services..... | 22 |
| CONCLUSION | 24 |

INTRODUCTION

The aim of this lab is to enhance our knowledge in terms of concepts and technologies for virtualization techniques. These techniques are used in IT environments that support the provisioning of novel software systems. These environments are typically dynamic and distributed. We will have to test 2 virtualization environments : VirtualBox and OpenStack. We are going to create and manage virtual machines and discover in which way they are dynamic.

THEORETICAL PART

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

The main goal of this lab is to enhance our knowledge in terms of concepts and technologies for virtualization techniques. There are many solutions available to achieve virtualization, here, we will focus on VM and CT technologies.

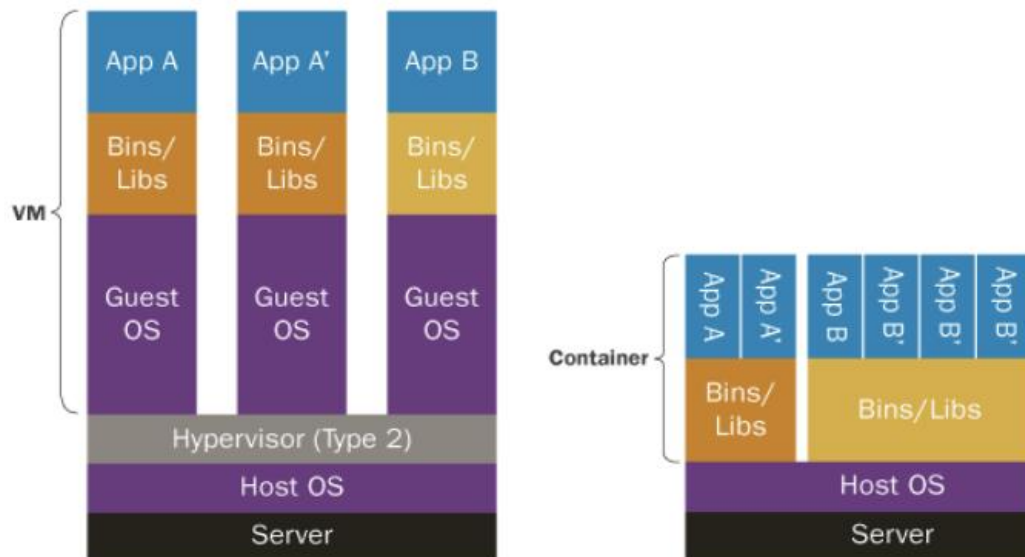


Figure 1: VM vs CT

The figure on the left illustrates a virtual machine (VM): using a hypervisor with a type 2 architecture. Whereas the figure on the right shows a container (CT) architecture.

The left image illustrates 3 VMs running on a single physical server. Each VM runs an application. We can notice the presence of 2 instances of "A" application and only one instance of "B" application.

This image is separated into 2 parts. First, the lower part, which represents the physical server (including its CPU, memory, network, disk resources), the OS and the hypervisor layer. All these layers are common and shared between the VMs in the 2nd image.

The upper part of the image shows three VMs that operate independently, each one of them with its own OS, runtime environment and applications. Each VM is dependent on the hypervisor layer which shares the resources below through the hypervisor layer. The OS could be heterogeneous Windows, Ubuntu etc.

The right image shows a physical server running two containers. There is no hypervisor layer, nor a VM. Each container is runtime independent of the other but shares the same OS. The two containers also share the same physical resources that are provided through the OS layer.

The first, container 1, contains two instances of "A" application, and a second, container 2, which contains 4 instances of "B" application. The instances on each container use the same version of the runtime environment, but it is not the same version of the runtime between the two containers.

Thus, in a more general way it shows the architectural differences, indeed with VM we can manage different type of OS that we call guest OS which are controlled by the hyper Supervisor running on the host OS, each VM has its own hardware resources allocated. With VM we can switch from one VM to another without doing a system reboot, to use different apps inside the same VM or multiple instances of an app in different VM running independently of each other. On the other side, with CT we have only one type of OS running, we can run different instances of the same application isolated from each other in the same CT. CT don't need a hypervisor since they can share the same OS kernel with same libraries, same hardware resources.

| Virtualization type | VM | CT |
|---|---|---|
| virtualization cost | <p>VM size is very large and VM uses a lot of system resources in term memory.</p> <p>Memory size = Gb.</p> <p>Scalability = Supports some scaling however it will reach the host ressource limits quickly.</p> <p>Reduce expenses > less stack of servers.</p> | <p>While the size of container is very light, i.e. a few megabytes. While containers require very less memory.</p> <p>Memory size = Mb</p> <p>Scalability = Easy to scale due to their small size, we can limit their memory and CPU usage.</p> |
| Usage of CPU, memory and Network for a given application | <p>VM's are useful when we require all of OS resources to run various applications.</p> <p>For a given application you must run a copy of an operating system as well as a copy of all the hardware required for the system to run. There is many network configurations possible</p> | <p>While containers are useful when we are required to maximise, the running applications using minimal servers.</p> |
| Security | <p>VM is more secure due to a strong isolation. Indeed it is isolated from the host OS > Safe for experimenting and developing applications and more stable</p> | <p>While containers are less secure. Moderate isolation because they share the same host's kernel, it is a process isolation, if a library is corrupted it will affect all the system and other apps</p> |
| Performances | <p>VM takes minutes to run, due to large size. High response time (starts in minutes)</p> <p>We have to launch each kernel for each VM and load whole kernel.</p> | <p>Better starting time : containers take a few seconds to run. Low response time (starts in seconds)</p> <p>We launch only one single kernel compared to VM which means fast response time</p> |
| Tooling integration | <p>Continuous</p> <p>Use of preinstalled tools possible</p> | <p>Must install some tools</p> |

| | VM | CT |
|---|--|---|
| <p>From an application developer's point of view</p> <p>Dev > flexibilité > Emulation > Test sur différent OS ></p> | <p>Relocating an app running on a virtual machine can also be complicated. However it allows testing an application in different OS types.</p> | <p>Faster and more flexible than VM . easier and faster to deploy new containers with ready to use applications already integrated. It enables the use of various applications in the same physical host.</p> <p>Easier to move to the cloud</p> <p>Contain microservices</p> <p>It reduces environment variables</p> <p>Make it easy to reuse code</p> <p>easier to secure applications because it isolates services and applications from each other however an application can affect another by compromising the stability of the host kernel.</p> <p>To conclude CT are better from an application dev's point of view, indeed, to do the automatization of the configuration of an application (config, initialization of all variables, login) we use some Tools like Jira/Jenkins/Python script and it is easier to find same natively on CT while we have to install them on VM.</p> <p>CT are made for devOps, they are used in that purpose, they are available, already interfaced and well integrated and very useful to automate the an application's circle of life.</p> |
| <p>From an infrastructure administrator point of view</p> | <p>Better for management > manage all the VM with the Hypervisor ></p> <p>More secure > Better for complex applications and data requirements.</p> <p>Allows a good backup alternative.</p> | <p>Easier for the administrator for the support on continuous integration : some orchestrators can be used to manage the allowed ressources, however it can be challenging to manage a large number of CT, to see what is running and where.</p> |

Similarities and differences between the existing CT types

| | Docker | LXC | LXD | OpenVZ | rkt |
|-------------------------------|------------------|-------------------------|-----------------|--------------------------|------------------------|
| Virtualization technologies | OS level | OS level | OS level | OS level, hypervisor | OS level, hypervisor |
| Full-system container | No | Yes | Yes | Yes | No |
| App container | Yes | No | No | No | Yes |
| Container format | Docker container | Linux container | Linux container | Virtuozzo containers | appc, Docker container |
| Linux kernel patch necessary? | No | No | No | Stand-alone distribution | No |
| Programming language | Go | C, Python 3, Shell, Lua | Go | C | Go |

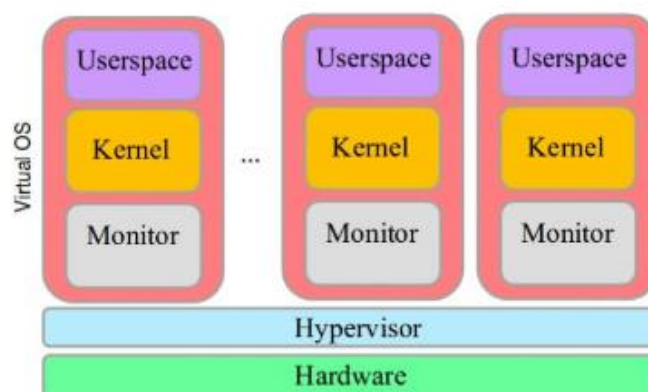
In the table above a couple of different container types are presented with some of the characteristics that set them apart. The gap in development and usage makes for a great variety of software and the features that each of them hold.

It is common to make a distinction between “full system containers” and “application containers”. OpenVZ and LXC are examples of full system container technologies. The goal of these programs is to offer a complete simulated OS to containers while not having to emulate the hardware needed on the host OS. Being containers, they are still more lightweight and have less overhead than a regular VM while still providing a good level of isolation.

Application containers are, as the name suggests, made for running a single application. They share characteristics with full system containers and could in theory be configured to enable all features of a “full” OS, but their usual job is that of a lighter program with a more specialized task.

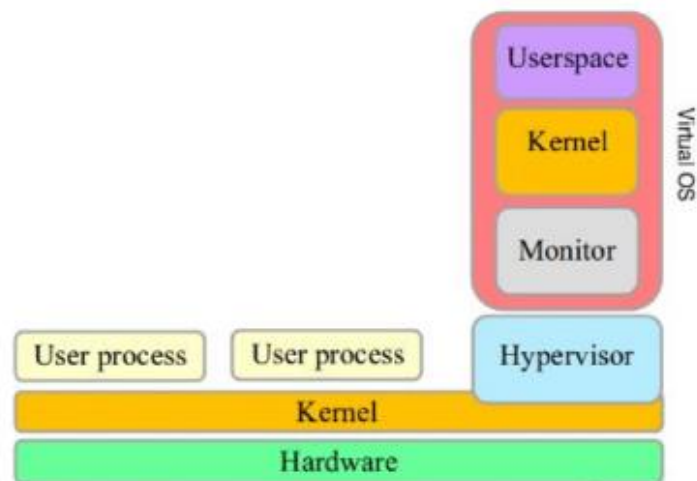
3. Similarities and differences between Type 1 & Type 2 of hypervisors’ architectures

A hypervisor works as a software which can create, manage and terminate VMs, and it enables the running of multiple OS on a single host machine. There are two types of architectures for the hypervisor: Type 1, often referred to as “bare-metal” or “hardware level” and Type 2, called “hosted” or “OS-level”.



Type 1 Architecture

Type 1, like the name suggests, works directly on the hardware level, and can therefore monitor every OS which is run on it. The OS and hypervisor are independent from each other, and the hypervisor's only task is to manage the resources between the operating systems. The main advantage with this type of architecture is security; if there is a problem on one or more of the VMs the other operating systems will not be affected. Another convenience is the availability; efficient hardware resource management enables an increased performance for the VMs.



Type 2 Architecture

The type 2 architecture runs on the operating system of the machine, it therefore depends on the host OS for all operations that are carried out on the VMs. This setup allows a better control of the parameters of the VM by the host OS but it also raises some drawbacks. The performance is comparably lower than that of the type 1 architecture because all requests from the VM needs to be processed by the hypervisor and, consequently, the base OS. Type 2 architectures also lack the advantages mentioned for the type 1 setups, as a problem in the base OS will affect the entire system overhead.

PRACTICAL PART

2. Tasks related to objectives 4 and 5

2.1 First part: Creating and configuring a VM

In this part the VirtualBox hypervisor in NAT mode is used to connect a VM to the network. The bridge mode will be used in the second part of this project, with an OpenStack hypervisor.

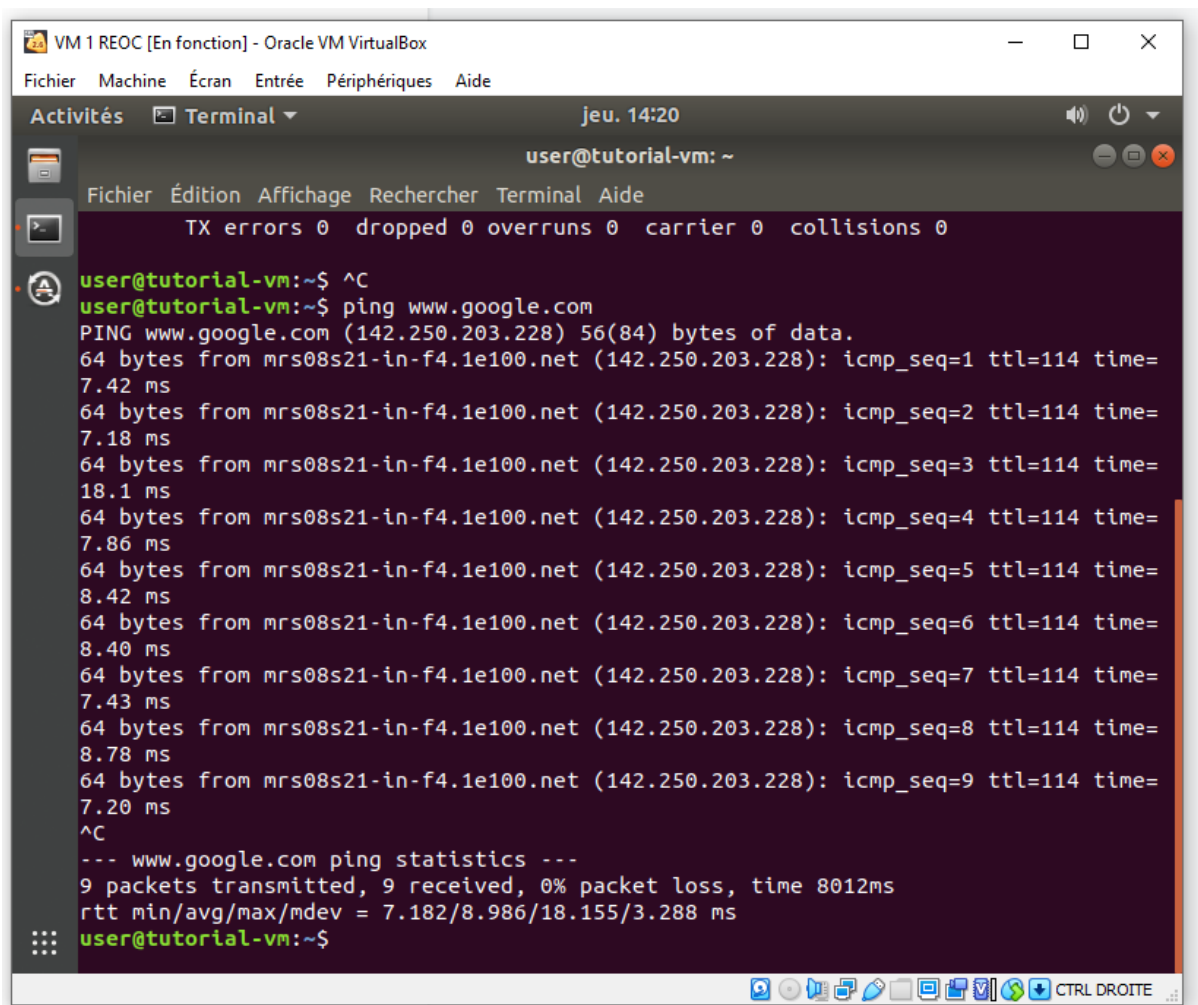
Using the wizard provided by VirtualBox we simply create a VM type Linux 64 bits, giving it a RAM of 512 MB and 1 core CPU.

2.2 Testing the VM connectivity

Using the ifconfig command line in the VM Linux, or ipconfig for the host machine we can retrieve the localhost and VM's IP address.

Then it is possible to check the connectivity with a ping test between the different points of communication: from the VM to the outside, a neighbour's host to our hosted VM and from our host to our hosted VM.

The ping test is working perfectly fine from the VM to the outside meaning that our VM is connected to the public internet. However, it doesn't work from a neighbour's host to our hosted VM and from our host to our hosted VM, indeed we are not in the same local area network, as we are applying a NAT to permit communication our VM doesn't have its own public IP address. That is why a neighbour PC and the host can't ping the VM, they have no way to address the VM. To fix this problem it will be necessary to use the port forwarding technique using the physical network from the host.

A screenshot of a VirtualBox terminal window titled "VM 1 REOC [En fonction] - Oracle VM VirtualBox". The terminal shows a Linux prompt "user@tutorial-vm: ~". The user enters "ping www.google.com". The output shows 9 successful ping packets with varying response times (7.20 ms to 18.1 ms). The statistics at the bottom indicate "9 packets transmitted, 9 received, 0% packet loss, time 8012ms" and "rtt min/avg/max/mdev = 7.182/8.986/18.155/3.288 ms".

```
user@tutorial-vm: ~  
TX errors 0 dropped 0 overruns 0 carrier 0 collisions 0  
user@tutorial-vm:~$ ^C  
user@tutorial-vm:~$ ping www.google.com  
PING www.google.com (142.250.203.228) 56(84) bytes of data.  
64 bytes from mrs08s21-in-f4.1e100.net (142.250.203.228): icmp_seq=1 ttl=114 time=  
7.42 ms  
64 bytes from mrs08s21-in-f4.1e100.net (142.250.203.228): icmp_seq=2 ttl=114 time=  
7.18 ms  
64 bytes from mrs08s21-in-f4.1e100.net (142.250.203.228): icmp_seq=3 ttl=114 time=  
18.1 ms  
64 bytes from mrs08s21-in-f4.1e100.net (142.250.203.228): icmp_seq=4 ttl=114 time=  
7.86 ms  
64 bytes from mrs08s21-in-f4.1e100.net (142.250.203.228): icmp_seq=5 ttl=114 time=  
8.42 ms  
64 bytes from mrs08s21-in-f4.1e100.net (142.250.203.228): icmp_seq=6 ttl=114 time=  
8.40 ms  
64 bytes from mrs08s21-in-f4.1e100.net (142.250.203.228): icmp_seq=7 ttl=114 time=  
7.43 ms  
64 bytes from mrs08s21-in-f4.1e100.net (142.250.203.228): icmp_seq=8 ttl=114 time=  
8.78 ms  
64 bytes from mrs08s21-in-f4.1e100.net (142.250.203.228): icmp_seq=9 ttl=114 time=  
7.20 ms  
^C  
--- www.google.com ping statistics ---  
9 packets transmitted, 9 received, 0% packet loss, time 8012ms  
rtt min/avg/max/mdev = 7.182/8.986/18.155/3.288 ms  
user@tutorial-vm:~$
```

Ping test from VM to the outside : workin

```

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.

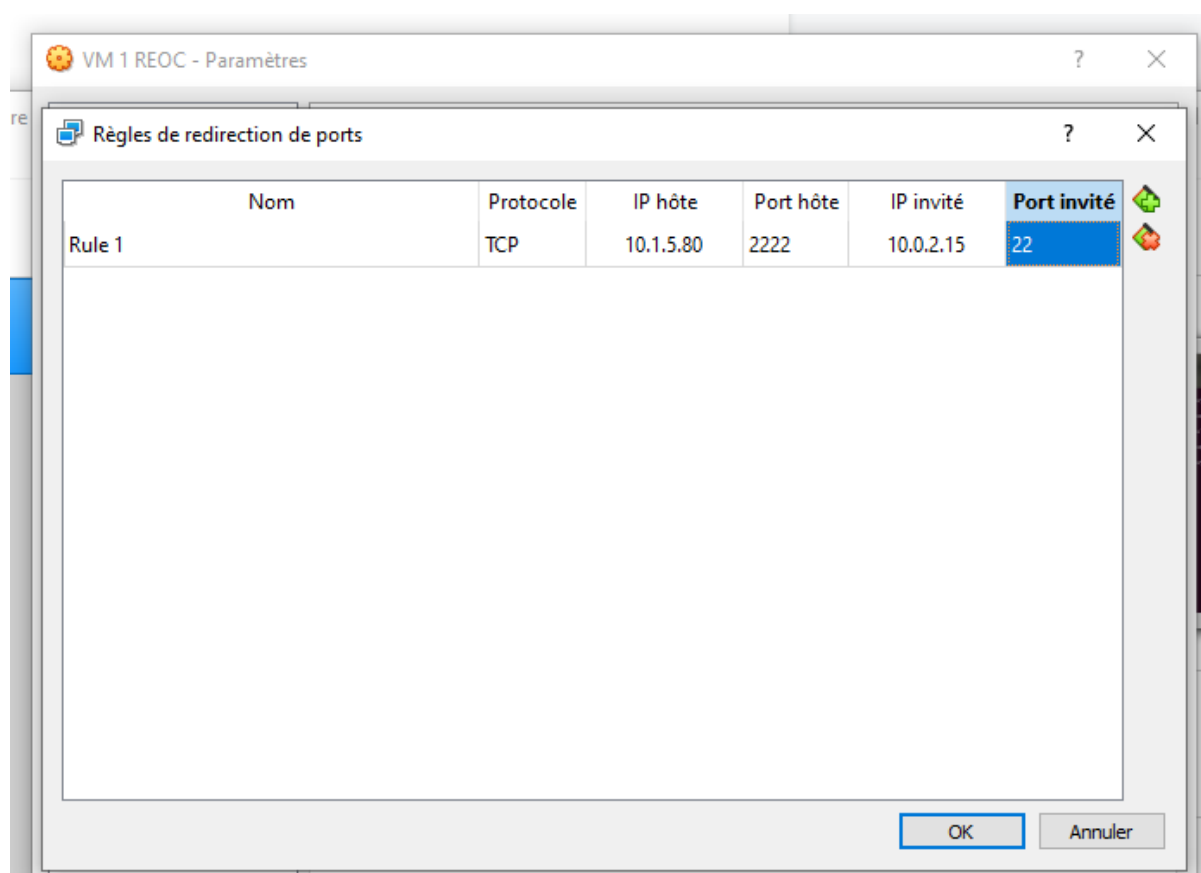
Statistiques Ping pour 10.0.2.15:
    Paquets : envoyés = 4, reçus = 0, perdus = 4 (perte 100%),
C:\Users\geneau>

```

Ping test from host to hosted VM is not working

2.3 Setting up the “missing” connectivity

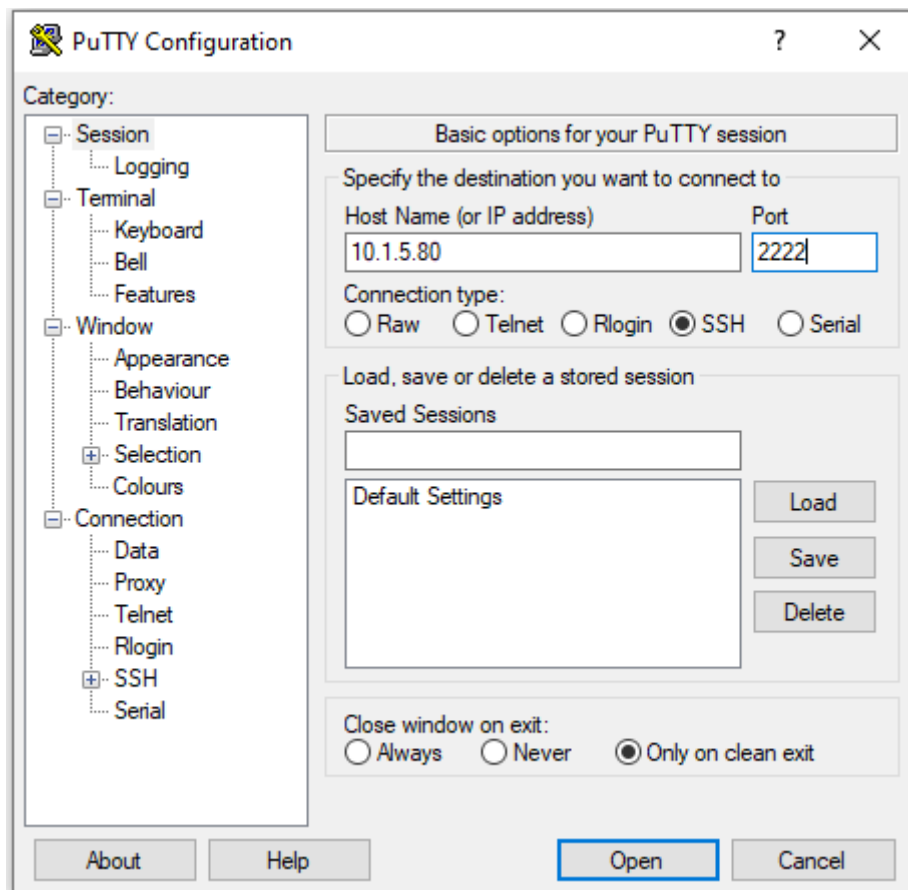
To solve the issue identified in the previous part, we connect from the exterior to the VM using a SSH application with a port forwarding technique.



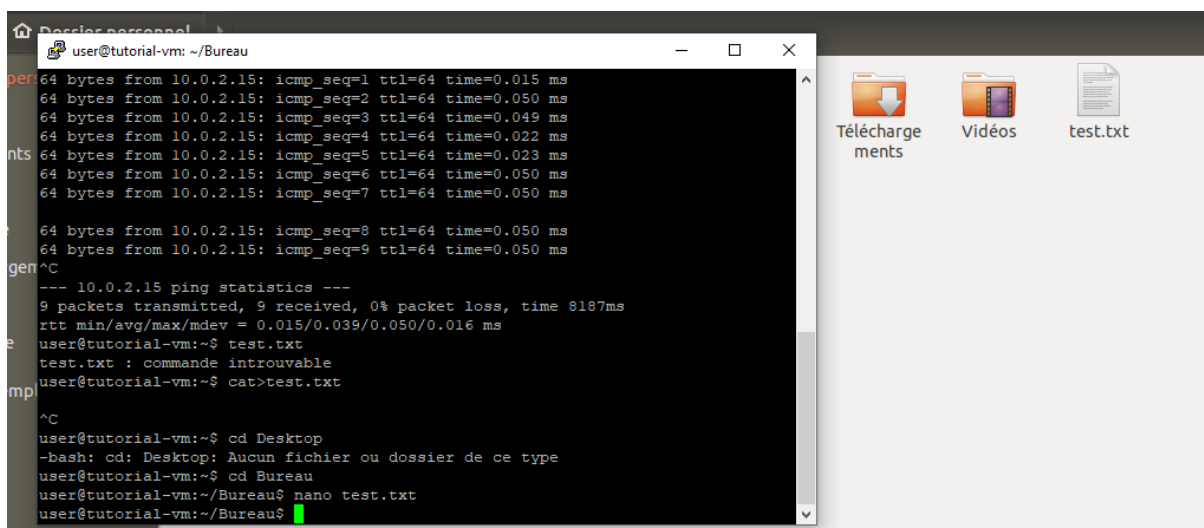
Port forwarding configuration

We can use VirtualBox to set a new network rule in the NAT network management part, in order to set the Port forwarding method. When an outside host tries to connect to our VM, the data is transferred via the port 2222 of the host's IP address 10.1.5.80 to the port 22 of the VM. With Putty application we could connect to the VM via the host.

A full two-way communication between the VM and the Internet is now established.



PuTTY configuration



Port forwarding result : txt file creation with PuTTY

2.4 Duplicating the VM

To make a clone of the VM that will contain all applications and settings installed on the original, we have several options. With the windows command it is possible to do a duplication, but this will clone the hardware disk only. To make the cloned VM we then create a new VM and associate it with the copy of the disk. It is also possible to use the GUI of VirtualBox by right clicking on the already existing VM and selecting "clone". This method clones the VM in its entirety, not only the disk. This operation also requires the VM to be shut down completely.

```
Microsoft Windows [version 10.0.19042.1348]
(c) Microsoft Corporation. Tous droits réservés.

C:\Users\geneau>"C:\Program Files\Oracle\VirtualBox\VirtualBox.exe" clonemedium "U:\Windows\Bureau\TP CLOUD\disk.vmdk" "
U:\Windows\Bureau\TPREOC\disk-copy2.vmdk"
0%...10%...20%...30%...40%...50%...60%...70%...80%...90%...100%
Clone medium created in format 'VMDK'. UUID: 17dc821c-6dd2-442a-96f3-8884004fef88
```

Command Line for VM duplication

2.5 Docker containers provisioning

In this step we're deploying the target Docker environment over the VM from VirtualBox as this will ensure that we have all the admin privileges. Provisioning containers over VMs is technically possible but remains rare in practice; containers are usually deployed bare-metal on physical hosts like in the case of VMs. To allocate a container we start by downloading an Ubuntu image with the command `sudo docker pull ubuntu` and then we execute an instance based on this image:

```
Fichier Édition Affichage Rechercher Terminal Aide
user@tutorial-vm:~$ sudo docker pull ubuntu
[sudo] Mot de passe de user :
Using default tag: latest
latest: Pulling from library/ubuntu
d72e567cc804: Pull complete
0f3630e5ff08: Pull complete
b6a83d81d1f4: Pull complete
Digest: sha256:bc2f7250f69267c9c6b66d7b6a81a54d3878bb85f1ebb5f951c896d13e6ba537
Status: Downloaded newer image for ubuntu:latest
user@tutorial-vm:~$ sudo docker run --name ct1 -it ubuntu
root@ebd3ec6ef2ea:/#
root@ebd3ec6ef2ea:/# sudo apt-get -y udate && apt-get -y install net-tools iputils-ping
bash: sudo: command not found
root@ebd3ec6ef2ea:/# $ sudo apt-get -y udate && apt-get -y install net-tools iputils-ping
bash: $: command not found
root@ebd3ec6ef2ea:/# ^C
root@ebd3ec6ef2ea:/# cd ..
root@ebd3ec6ef2ea:/# sudo apt-get -y udate && apt-get -y install net-tools iputils-ping
bash: sudo: command not found
root@ebd3ec6ef2ea:/# apt-get -y udate && apt-get -y install net-tools iputils-ping
E: Invalid operation udate
root@ebd3ec6ef2ea:/# apt-get -y update && apt-get -y install net-tools iputils-ping
Get:1 http://archive.ubuntu.com/ubuntu focal InRelease [265 kB]
Get:2 http://archive.ubuntu.com/ubuntu focal-updates InRelease [111 kB]
Get:3 http://archive.ubuntu.com/ubuntu focal-backports InRelease [98.3 kB]
```

Get an Ubuntu image and execution of an instance

As the figure shows there is no need to put the "\$ sudo" in the command prompt as we are in the docker already and it is always rooted. This part of the command is therefore omitted. To test the connectivity of this container we first need to install the necessary connectivity testing tools:

```
nu/perl/5.30 /usr/share/perl/5.30 /usr/local/lib/site_perl /usr/lib/x86_64-linux-gnu/perl-base) at /usr/share/perl5/Debconf/FrontEnd/Readline.pm line 7.)
debconf: falling back to frontend: Teletype
Setting up iputils-ping (3:20190709-3) ...
Processing triggers for libc-bin (2.31-0ubuntu9.1) ...
root@ebd3ec6ef2ea:/# ifconfig
eth0: flags=4163<UP,BROADCAST,RUNNING,MULTICAST> mtu 1500
    inet 172.17.0.2 netmask 255.255.0.0 broadcast 172.17.255.255
    ether 02:42:ac:11:00:02 txqueuelen 0 (Ethernet)
    RX packets 5132 bytes 16781569 (16.7 MB)
    RX errors 0 dropped 0 overruns 0 frame 0
    TX packets 4632 bytes 255114 (255.1 KB)
    TX errors 0 dropped 0 overruns 0 carrier 0 collisions 0

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

root@ebd3ec6ef2ea:/#
```

Docker IP address

Container and testing tools have been set up, we then proceed to test the different aspects of the container communication system.

```

TX errors 0 dropped 0 overruns 0 carrier 0 collisions 0

root@ffb7379f479f:/# ping www.google.com
PING www.google.com (142.250.203.228) 56(84) bytes of data.
64 bytes from mrs08s21-in-f4.1e100.net (142.250.203.228): icmp_seq=1 ttl=113 time=7.83 ms
64 bytes from mrs08s21-in-f4.1e100.net (142.250.203.228): icmp_seq=2 ttl=113 time=8.46 ms
64 bytes from mrs08s21-in-f4.1e100.net (142.250.203.228): icmp_seq=3 ttl=113 time=9.26 ms
64 bytes from mrs08s21-in-f4.1e100.net (142.250.203.228): icmp_seq=4 ttl=113 time=7.82 ms
64 bytes from mrs08s21-in-f4.1e100.net (142.250.203.228): icmp_seq=5 ttl=113 time=8.74 ms
64 bytes from mrs08s21-in-f4.1e100.net (142.250.203.228): icmp_seq=6 ttl=113 time=8.75 ms
64 bytes from mrs08s21-in-f4.1e100.net (142.250.203.228): icmp_seq=7 ttl=113 time=8.12 ms
64 bytes from mrs08s21-in-f4.1e100.net (142.250.203.228): icmp_seq=8 ttl=113 time=8.13 ms
^C
--- www.google.com ping statistics ---
8 packets transmitted, 8 received, 0% packet loss, time 7013ms
rtt min/avg/max/mdev = 7.823/8.386/9.255/0.471 ms
root@ffb7379f479f:/#

```

Ping test from Docker to an internet resource : working

```

root@ffb7379f479f:/# ping 10.1.5.80
PING 10.1.5.80 (10.1.5.80) 56(84) bytes of data.
64 bytes from 10.1.5.80: icmp_seq=1 ttl=126 time=0.964 ms
64 bytes from 10.1.5.80: icmp_seq=2 ttl=126 time=1.45 ms
64 bytes from 10.1.5.80: icmp_seq=3 ttl=126 time=1.48 ms
64 bytes from 10.1.5.80: icmp_seq=4 ttl=126 time=1.43 ms
64 bytes from 10.1.5.80: icmp_seq=5 ttl=126 time=1.50 ms
^C
--- 10.1.5.80 ping statistics ---
5 packets transmitted, 5 received, 0% packet loss, time 4006ms
rtt min/avg/max/mdev = 0.964/1.365/1.501/0.202 ms
root@ffb7379f479f:/#

```

Figure 11: Ping test from Docker to the host : working

```

user@tutorial-vm:~$ ping 127.0.0.1
PING 127.0.0.1 (127.0.0.1) 56(84) bytes of data.
64 bytes from 127.0.0.1: icmp_seq=1 ttl=64 time=0.041 ms
64 bytes from 127.0.0.1: icmp_seq=2 ttl=64 time=0.060 ms
64 bytes from 127.0.0.1: icmp_seq=3 ttl=64 time=0.059 ms
64 bytes from 127.0.0.1: icmp_seq=4 ttl=64 time=0.025 ms
64 bytes from 127.0.0.1: icmp_seq=5 ttl=64 time=0.062 ms
64 bytes from 127.0.0.1: icmp_seq=6 ttl=64 time=0.061 ms
^C
--- 127.0.0.1 ping statistics ---
6 packets transmitted, 6 received, 0% packet loss, time 5101ms
rtt min/avg/max/mdev = 0.025/0.051/0.062/0.015 ms
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=0.050 ms
64 bytes from 172.17.0.2: icmp_seq=2 ttl=64 time=0.041 ms
64 bytes from 172.17.0.2: icmp_seq=3 ttl=64 time=0.047 ms
64 bytes from 172.17.0.2: icmp_seq=4 ttl=64 time=0.071 ms
64 bytes from 172.17.0.2: icmp_seq=5 ttl=64 time=0.070 ms
^C

```

Ping test from VM to the docker : working

As shown on the photos, the Docker IP address displays without problem and we managed to access an Internet resource and the host from inside the container. While the VM relies on a NAT to communicate with the outside, the container uses a bridge mode: the Docker is virtually connected to the local network of the host machine, and can ping the host in spite of not being on the same addressing space. As the container has connectivity to the VM through bridge mode and to the Internet through the VM, the Docker can therefore reach the internet without problem.

2.5.1 Multiple containers

For this part we first execute a new instance (CT2) of the ubuntu Docker with the command `sudo docker run --name ct2 -p 2223:22 -it ubuntu`. Then we install a text editor, nano, on the CT2 with the command `apt-get -y update && apt install nano`.

We then make a snapshot of the CT2 before stopping and terminating CT2:

```
user@tutorial-vm:~$ sudo docker ps -a
CONTAINER ID        IMAGE               NAMES             COMMAND           CREATED            STATUS
PORTS              NAMES
d63dd59d65b1       ubuntu             ct2               "bash"           2 minutes ago     Exited (0) 32 seconds ago
ffb7379f479f       ubuntu             ct1               "bash"           26 minutes ago    Exited (0) 3 minutes ago

user@tutorial-vm:~$ sudo docker commit d63dd59d65b1 images:01
sha256:8984afde6999f621b78ae7569ae28b7c2825dbc4d982c0571057f37cc48083d3
```

Snapshot of CT2

Then we execute a new instance named CT3 from the snapshot we just created:

As shown we still have nano installed, this is because it was executed from a snapshot of CT2, where nano was already installed - the snapshot included instructions and the program files for nano.

```
user@tutorial-vm:~$ sudo docker run --name ct3 -it images:01
root@8e95d4336cdc:/# nano
root@8e95d4336cdc:/# nano -v
root@8e95d4336cdc:/# nano --v
nano: option '--v' is ambiguous; possibilities: '--version' '--view'
Type 'nano -h' for a list of available options.
root@8e95d4336cdc:/# nano --version
GNU nano, version 4.8
(C) 1999-2011, 2013-2020 Free Software Foundation, Inc.
(C) 2014-2020 the contributors to nano
Email: nano@nano-editor.org   Web: https://nano-editor.org/
Compiled options: --disable-libmagic --enable-utf8
root@8e95d4336cdc:/#
```

CT3 instance and nano testing

```
Fichier  Édition  Affichage  Recherche  Terminal  Aide
GNU nano 2.9.3 myDocker.dockerfile

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

Dockerfile creation

```
user@tutorial-vm:~$ nano myDocker.dockerfile
user@tutorial-vm:~$ sudo docker build -t images:02 -f myDocker.dockerfile .
Sending build context to Docker daemon 139.9MB
Step 1/4 : FROM ubuntu
--> ba6accedd29
Step 2/4 : RUN apt update -y
--> Running in f67629947258

WARNING: apt does not have a stable CLI interface. Use with caution in scripts.

Get:1 http://security.ubuntu.com/ubuntu focal-security InRelease [114 kB]
Get:2 http://archive.ubuntu.com/ubuntu focal InRelease [265 kB]
```

Building of the image with the previous Dockerfile

```
Successfully built 2578a578bd3d
Successfully tagged images:02
user@tutorial-vm:~$ sudo docker images
REPOSITORY          TAG                 IMAGE ID            CREATED             SIZE
images              02                 2578a578bd3d       10 seconds ago     106MB
images              01                 8984afde6999       12 minutes ago     106MB
<none>              <none>             7b08fd192437       18 minutes ago     106MB
ubuntu              latest             ba6accce9d29       5 weeks ago        72.8MB
ubuntu              <none>             597ce1600cf4       7 weeks ago        72.8MB
user@tutorial-vm:~$
```

Image creation using Dockerfile

3. Tasks related to objectives 6 and 7

We connected to OpenStack using the VPN. During the creation of the VM, it didn't work in the first part because there isn't a private network, and we don't have the right to create a public IP address with INSA network, thus we had to create it in order to be able to create the VM.

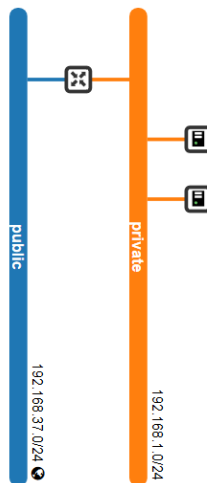
Erreur : N'a pas pu effectuer l'opération demandée sur l'instance "Instance1", l'instance a un statut d'erreur: Veuillez essayer à nouveau ultérieurement [Error : Exceeded maximum number of retries. Exceeded max scheduling attempts 3 for instance 55218859-b569-40c2-bdfb-73c7b2a67a1c. Last exception: Binding failed for port d16181ca-4200-410c-bc73-2af274b55c56, please check neutron logs for more information.]

Error during VM creation due to the public network used

3.2 Connectivity test

After having authorize the ping and SSH traffic managing the rules we can now test with pings the VM connectivity. However, it is not working, and it is normal, we allowed the ping with ICMP and the SSH but we still need an floating IP, we have to add a virtual router to be able to access public network with the VM's private network. We end up with a VM with an associated IP address.

After the creation of the router we allocate the floating IP to the VM which is part of the public network, thus we can now access the VM from the public network.



Network topology

We will now test the connectivity of our VM associated with the floating IP and the router as a gateway between the private and the public network.

```

alpine-node:~# ping www.google.com
PING www.google.com (142.251.37.164): 56 data bytes
64 bytes from 142.251.37.164: seq=0 ttl=114 time=7.539 ms
64 bytes from 142.251.37.164: seq=1 ttl=114 time=6.963 ms
64 bytes from 142.251.37.164: seq=2 ttl=114 time=6.722 ms
64 bytes from 142.251.37.164: seq=3 ttl=114 time=6.794 ms
64 bytes from 142.251.37.164: seq=4 ttl=114 time=7.091 ms
64 bytes from 142.251.37.164: seq=5 ttl=114 time=6.841 ms
64 bytes from 142.251.37.164: seq=6 ttl=114 time=6.912 ms
64 bytes from 142.251.37.164: seq=7 ttl=114 time=6.775 ms
64 bytes from 142.251.37.164: seq=8 ttl=114 time=6.747 ms
64 bytes from 142.251.37.164: seq=9 ttl=114 time=6.745 ms
^C
--- www.google.com ping statistics ---
10 packets transmitted, 10 packets received, 0% packet loss
round-trip min/avg/max = 6.722/6.912/7.539 ms
alpine-node:~#
  
```

Ping test from VM to exterior : working

```

alpine-node:~# ping 10.1.5.86
PING 10.1.5.86 (10.1.5.86): 56 data bytes
64 bytes from 10.1.5.86: seq=0 ttl=126 time=2.242 ms
64 bytes from 10.1.5.86: seq=1 ttl=126 time=1.098 ms
64 bytes from 10.1.5.86: seq=2 ttl=126 time=1.101 ms
64 bytes from 10.1.5.86: seq=3 ttl=126 time=1.217 ms
^C
--- 10.1.5.86 ping statistics ---
4 packets transmitted, 4 packets received, 0% packet loss
round-trip min/avg/max = 1.098/1.414/2.242 ms
alpine-node:~#
  
```

Ping test from VM to host : working

```

Envoi d'une requête 'Ping' 192.168.37.119 avec 32 octets de données :
Réponse de 192.168.37.119 : octets=32 temps=2 ms TTL=62
Réponse de 192.168.37.119 : octets=32 temps=1 ms TTL=62
Réponse de 192.168.37.119 : octets=32 temps=1 ms TTL=62
Réponse de 192.168.37.119 : octets=32 temps=1 ms TTL=62

Statistiques Ping pour 192.168.37.119:
  Paquets : envoyés = 4, reçus = 4, perdus = 0 (perte 0%),
  Durée approximative des boucles en millisecondes :
    Minimum = 1ms, Maximum = 2ms, Moyenne = 1ms
  
```

```

U:\>ssh root@192.168.37.119
The authenticity of host '192.168.37.119 (192.168.37.119)' can't be established.
ECDSA key fingerprint is SHA256:2ipwDaAMhfg+fzvKcwyjtsnSdWyrbDIZ4i38WGwawnY.
Are you sure you want to continue connecting (yes/no)? y
Please type 'yes' or 'no': yes
Warning: Permanently added '192.168.37.119' (ECDSA) to the list of known hosts.
root@192.168.37.119's password: 
Permission denied, please try again.
root@192.168.37.119's password:

```

Ping test from host to VM : not working even with SSH connection using windows CMD

We must use a complicated command to use SSH with alpine we have to go on the file etc/ssh/ using the command nano sshd_config and then find the paramet PermiRootLogin and change it to : YES, and we also need to erase the # symbol.

```

# Ciphers and keying
#RekeyLimit default none

# Logging
#SyslogFacility AUTH
#LogLevel INFO

# Authentication:

#LoginGraceTime 2m
#PermitRootLogin yes
#StrictModes yes
#MaxAuthTries 6
#MaxSessions 10

alpine-node:/etc/ssh# #/etc/init.d/sshd-restart
alpine-node:/etc/ssh# #/etc/init.d/sshd_restart
alpine-node:/etc/ssh# #/etc/init.d/sshd restart
alpine-node:/etc/ssh# #/etc/init.d/sshd restart
alpine-node:/etc/ssh# ../init.d/sshd restart
* Stopping sshd ...
* Starting sshd ...
alpine-node:/etc/ssh#

```

Parameter to change for SSH connection and restart of the VM to apply the change

And after testing the SSH connection we can observe that it is working perfectly fine now.

```

U:\>ssh root@192.168.37.119
root@192.168.37.119's password:
Welcome to Alpine!

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

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

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

alpine-node:~#

```

SSH connection working

3.3 Snapshot, restore and resize a VM

OpenStack allows users to resize a VM from the Instances tab, even while it is running or shutdown. However, we could not test the resize function due to access right restriction with our student OpenStack session.

4. Tasks related to objectives 8 and 9

4.1 OpenStack client installation

First, it is necessary to authorize the traffic, indeed we will use ports between 50000 and 50050, they are the ones allowed by the CSN. For that we must set a new network rule :

Ajouter une règle

Règle ^{*}

Règle TCP personnalisée

Description [?]

Direction

Entrée

Ouvrir ^{*}

Port

Port [?]

50000

Distant ^{*} [?]

CIDR

CIDR [?]

0.0.0.0/0

Description :

Les règles définissent quel t instances associées au grou sécurité se compose de trois

Règle : Vous pouvez spécifi ou utiliser des règles person Règle TCP Personnalisée, F Règle ICMP personnalisée.

Ouvrir Port / Plage de Port UDP, vous pouvez choisir d' une plage de ports. En sélectionner ports", vous aurez l'espace p le port de fin pour la plage c ICMP, vous devez en revanc code dans les espaces prop

Distant : Vous devez spécif par cette règle. Vous pouvez d'un bloc d'adresses IP (CIC groupe source (groupe de s sécurité comme source auto de ce groupe à accéder à n au travers de cette règle.

New TCP rule to allow communication with the port 50000 (In and Out)

We have to install the OpenStack command line client to be able to use the remote REST operations.

```
total 8
drwxr-xr-x  2 user user 4096 oct.  4 2019 .
drwxr-xr-x 15 user user 4096 oct. 16 14:44 ..
user@tutorial-vn:~/Bureau$ cd ..
user@tutorial-vn:~$ cd Téléchargements/
user@tutorial-vn:~/Téléchargements$ ls
SISS-Cloud-B23-openrc.sh
user@tutorial-vn:~/Téléchargements$ SISS-Cloud-B23-openrc.sh
SISS-Cloud-B23-openrc.sh : commande introuvable
user@tutorial-vn:~/Téléchargements$ SISS-Cloud-B23-openrc.sh
SISS-Cloud-B23-openrc.sh : commande introuvable
user@tutorial-vn:~/Téléchargements$ source SISS-Cloud-B23-openrc.sh
Please enter your OpenStack Password for project SISS-Cloud-B23 as user geneau:
user@tutorial-vn:~/Téléchargements$ openstack
(openstack) help

Shell commands (type help <topic>):
=====
cmdenvironment  exit  history  py          quit  save  shell    show
edit           help load  pyscript run    set   shortcuts

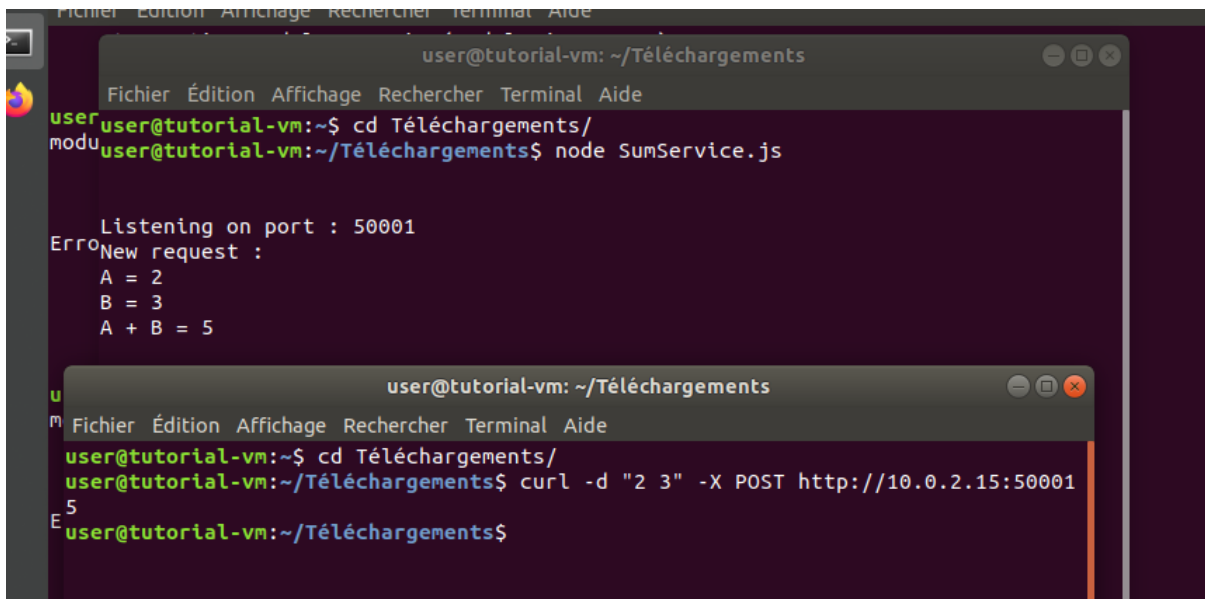
Application commands (type help <topic>):
=====
access token create                                network rbac list
```

Download of RC v3, execution of the file and help display

4.2. Web 2-tier application topology and specification

In this part we just have to install the required runtime and cURL to start and use micro-services, because micro-services have been already written, we don't need to code them.

We will try it with an Ubuntu VM. It is necessary to install firefox to charge each micro-service, and then we have to change the CalculatorService lines using the command nano in order to change the port number and the IP address for the VM's IP address.

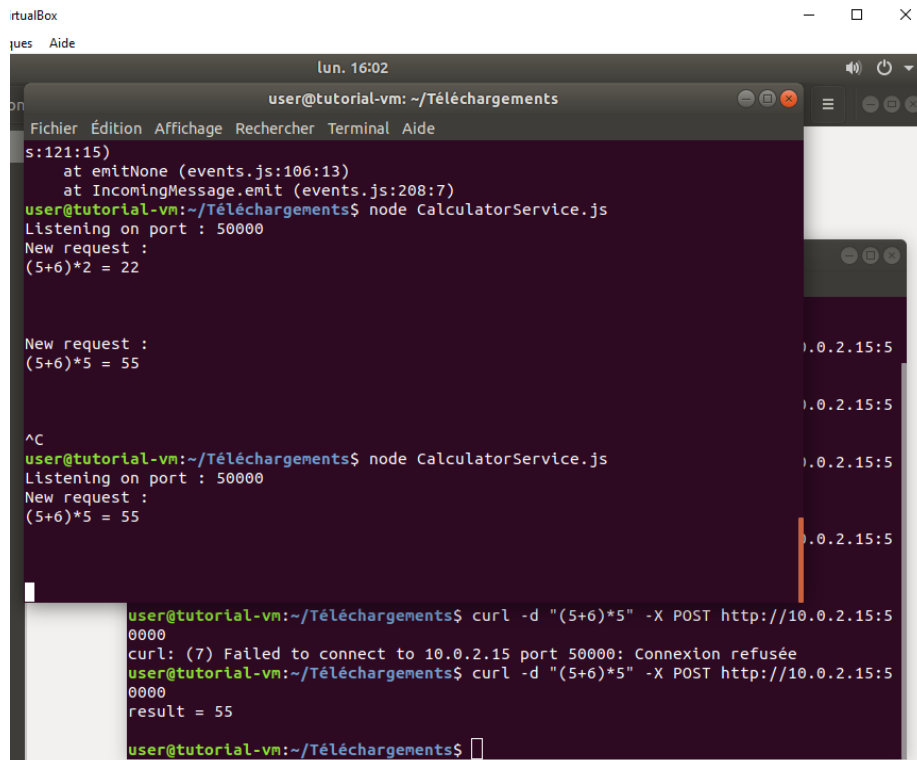


```
user@tutorial-vm: ~/Téléchargements
Fichier Édition Affichage Rechercher Terminal Aide
user@tutorial-vm:~$ cd Téléchargements/
user@tutorial-vm:~/Téléchargements$ node SumService.js

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

user@tutorial-vm: ~/Téléchargements
Fichier Édition Affichage Rechercher Terminal Aide
user@tutorial-vm:~$ cd Téléchargements/
user@tutorial-vm:~/Téléchargements$ curl -d "2 3" -X POST http://10.0.2.15:50001
5
user@tutorial-vm:~/Téléchargements$
```

Testing the sum service with two terminals



```
lun. 16:02
user@tutorial-vm: ~/Téléchargements
Fichier Édition Affichage Rechercher Terminal Aide
s:121:15)
  at emitNone (events.js:106:13)
  at IncomingMessage.emit (events.js:208:7)
user@tutorial-vm:~/Téléchargements$ node CalculatorService.js
Listening on port : 50000
New request :
(5+6)*2 = 22

New request :
(5+6)*5 = 55

^C
user@tutorial-vm:~/Téléchargements$ node CalculatorService.js
Listening on port : 50000
New request :
(5+6)*5 = 55

user@tutorial-vm:~/Téléchargements$ curl -d "(5+6)*5" -X POST http://10.0.2.15:50000
curl: (7) Failed to connect to 10.0.2.15 port 50000: Connexion refusée
user@tutorial-vm:~/Téléchargements$ curl -d "(5+6)*5" -X POST http://10.0.2.15:50000
result = 55
user@tutorial-vm:~/Téléchargements$
```

Testing the Calculator service

4.3 Deploy the Calculator application on OpenStack

We just have to associate one floating IP to the Calculator micro-service

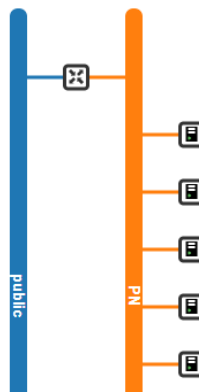
We created an independent VM for each micro-service as we can see below :

Affichage de 7 éléments

| <input type="checkbox"/> | Instance Name | Image Name | IP Address | Flavor | Key Pair | Status | | Availability Zone |
|--------------------------|----------------|------------|-------------------------------|--------|----------|--------|--|-------------------|
| <input type="checkbox"/> | VM Div | VM 3 test | 192.168.0.12 | tiny | - | Active | | nova |
| <input type="checkbox"/> | VM Mul | VM 3 test | 192.168.0.95 | tiny | - | Active | | nova |
| <input type="checkbox"/> | VM Sub | VM 3 test | 192.168.0.14 | tiny | - | Active | | nova |
| <input type="checkbox"/> | VM Sum | VM 3 test | 192.168.0.61 | tiny | - | Active | | nova |
| <input type="checkbox"/> | VM Calculator1 | VM 3 test | 192.168.0.133, 192.168.37.171 | tiny | - | Active | | nova |

VM dedicated to each micro-service

Each VM has to be on the same private network, we end up with that final network topology :



With our application the client will communicate with the Calculator service which will redirect all the operation to each micro-service dedicated calling the different web services when it is needed.

We will have to change again the CalculatorService.js file in order to apply the right IP address for each VM running the web services and also to change the port between 50000 to 50050. What is more, we also have the change the floating IP address of the CalculatorService VM.

```

Connected (encrypted) to: QEMU (instance-0000339d)
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://192.168.0.61:50001';
const SUB_SERVICE_IP_PORT = 'http://192.168.0.14:50002';
const MUL_SERVICE_IP_PORT = 'http://192.168.0.95:50003';
const DIV_SERVICE_IP_PORT = 'http://192.168.0.12:50004';

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);
    }
  }
}

File Name to Write: CalculatorService.js_
^G Get Help      M-D DOS Format  M-A Append      M-B Backup File
^C Cancel        M-M Mac Format  M-P Prepend     ^T To Files
  
```

Final configured file

Then we install the sync-request in the CalculatorService VM to be able to communicate with the exterior.

Then we tested our application successfully using cURL with a windows command prompt :

```
U:\>curl -d "(5+2)+7" -X POST http://192.168.37.171:50000
result = 14

U:\>curl -d "(5+2)*2" -X POST http://192.168.37.171:50000
result = 14

U:\>curl -d "(4+6)*2/4" -X POST http://192.168.37.171:50000
result = 5

U:\>curl -d "(4+7-1)*2/4" -X POST http://192.168.37.171:50000
result = 5

U:\>
```

```
alpine-node:~# node CalculatorService.js
Listening on port : 50000
New request :
(5+2)*2 = 14

New request :
(4+6)*2/4 = 5

New request :
(4+7-1)*2/4 = 5
```

Several test command and the result with the VM CalculatorService

```
connecting to homepage: Raas.IT (192.168.1.137)
MulService.js 100% |*****| 746 0:00:00 ETA
alpine-node:~# node MulService.js
Listening on port : 50003
New request :
A = 7
B = 2
A * B = 14

New request :
A = 10
B = 2
A * B = 20

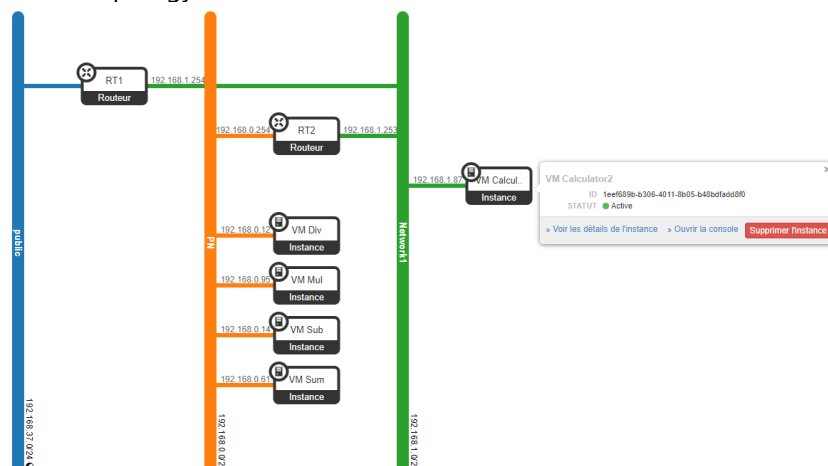
New request :
A = 10
B = 2
A * B = 20
```

VM MulService responding correctly

5. Tasks related to objectives 10 and 11

5.1 Configuration of the topology and execution of the services

We changed the network topology as shown below :



Then we changed the calculator service's code to adapt the IP addresses. However, pinging another VM hosting calculating operations with the VM wasn't working and we could not use the service, indeed the new network just created is still unknown to the calculator VM.

```
U:\>curl -d "(4+7-1)*2/4" -X POST http://192.168.37.171:50000
curl: (7) Failed to connect to 192.168.37.171 port 50000: Timed out
U:\>
```

Impossible to use the service

In order to fix the communication problem between the main VM and the sub-operating VM we need to define a traffic route between the 2 sub-networks that we created previously to adapt our topology to the client's demand :

```
alpine-node:~# route add -net 192.168.0.0/24 gw 192.168.0.254
route: ioctl 0x890b failed: Network unreachable
alpine-node:~# route add -net 192.168.0.0/24 gw 192.168.0.254
route: ioctl 0x890b failed: Network unreachable
alpine-node:~# ip route
default via 192.168.1.254 dev eth0 metric 202
192.168.1.0/24 dev eth0 scope link src 192.168.1.87
alpine-node:~# route add -net 192.168.0.0/24 gw 192.168.1.253
alpine-node:~# ping 192.168.0.95
PING 192.168.0.95 (192.168.0.95): 56 data bytes
64 bytes from 192.168.0.95: seq=0 ttl=63 time=5.066 ms
64 bytes from 192.168.0.95: seq=1 ttl=63 time=1.599 ms
64 bytes from 192.168.0.95: seq=2 ttl=63 time=1.218 ms
64 bytes from 192.168.0.95: seq=3 ttl=63 time=1.152 ms
64 bytes from 192.168.0.95: seq=4 ttl=63 time=1.484 ms
^C
--- 192.168.0.95 ping statistics ---
5 packets transmitted, 5 packets received, 0% packet loss
round-trip min/avg/max = 1.152/2.103/5.066 ms
alpine-node:~#
```

Traffic route added and ping test from the Calculator VM to a sub-VM

It is now possible to execute the calculator service using cURL in a windows command prompt and it is working perfectly fine :

```
U:\>curl -d "4*2" -X POST http://192.168.37.171:50000
result = 8
U:\>
```

```
64 bytes from 192.168.0.95: seq=2 ttl=63 time=1.218 ms
64 bytes from 192.168.0.95: seq=3 ttl=63 time=1.152 ms
64 bytes from 192.168.0.95: seq=4 ttl=63 time=1.484 ms
^C
--- 192.168.0.95 ping statistics ---
5 packets transmitted, 5 packets received, 0% packet loss
round-trip min/avg/max = 1.152/2.103/5.066 ms
alpine-node:~# node CalculatorService.js
Listening on port : 50000
New request :
4*2 = 8
```

Testing the application using cURL : working

CONCLUSION

During this computer lab exercise, we have had the opportunity to familiarize us with the several concepts regarding virtualization techniques. We have seen and experimented with the characteristics of the different architectures, and we have been able to experience first-hand the measures and factors to take into consideration when setting up a virtual machine or container. In addition to this we have seen several management operations like snapshots, restore and backup and we have gotten to recognize their usefulness in administering the different virtualisation processes.

The difference in network connectivity methods between the two has also been elaborated, NAT and bridge modes being the most important. Furthermore, we have learned about the OpenStack software and how to use it to manage Docker containers, virtual machines and their network connectivity. We have also seen how this software's API can be used to automate the management operations described previously.

In brief, this project has given us a lot of hands-on experience with some of the methods of virtualization. Finally, it has shown the basic usage of such processes, laying the groundwork for being able to use it proficiently in future projects.