Zennaro Thomas
Licini Valentin
5ISS - B11

INSA TOULOUSE

Cloud practical works report

tiny.cc/TP_Cloud

Theoretical Part

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

Criteria	Virtual Machine	Container
Virtualization cost	expensive ; big core allocation when VM is running	cheaper ; only one OS to maintain
usage of CPU and network	heavy ; connected directly to the network ; big use of storage ; memory management	use only the required resources ; connected through a bridge ; memory management not good
Security of the app	ressources provisioning ; more isolated ; security control is present when the OS is attacked	no easy ressources provisioning ; no security provided as most applications run on a single OS
Performances	worse	better ; can virtually run anywhere
Tooling for continuous integration support	retrieving of data loss better ; continuous deployment is not possible and testing is not done frequently	continuous deployment is possible and testing with different apps is carried out ; Applications are easily portable as they are stored in virtual containers

User preferences

User	Container	Virtual Machine
Developers	continuous integration portability same environment	
Administrators		Security management network configuration

2) Similarities and differences between the existing CT types

CT Technologies	Application isolation and resources, from a multi-tenancy point of view	Containerization level	Tooling
LXC/LXD	Good management of memory use ; running of multiple isolated Linux virtual environments (VE) on a single control host ; It allows you to not only isolate applications, but even the entire OS	high level	REST API ; LXC offers a rich set of tools that are almost identical to your traditional Linux machine ; offers a comprehensive set of tools for creating, running, and managing LXC containers on the fly
Docker	provides a lightweight virtualization solution to run processes in isolation	high level (engine)	Docker containers aim to be even lighter weight in order to support the fast, highly scalable, deployment of applications with microservice architecture
Rocket	allows users to apply different configurations (like isolation parameters) ; rocket's architecture means that each pod executes directly in the classic Unix process model in a self-contained, isolated environment	high level	ensures portability: rkt is designed to run App Container Images (ACI) ; It is inherently secure, using signature verification and privilege separation by default
OpenVZ	Isolated applications but partial cryp, fluid memory use, only linux containers and they rely on one kernel only; so an error on one container shut the others.	high level	Allow for a light and resource efficient use of linux based containers. Optimized for server purpose.

4) Similarities and differences between Type 1 & Type 2 of hypervisors' architectures

Type 1 : Bare-metal

Type 1 hypervisors run directly on the hardware. It is completely independent from the Operating System (OS).

A Type 1 hypervisor provides excellent performance and stability since it does not run inside any operating system. The hypervisor is small as its main task is sharing and managing hardware resources between different operating systems

Type 1 hypervisors are an OS themselves, a very basic one on top of which you can run virtual machines. If there is a problem with one of the VM, it doesn't affect the other VMs.

Type 2 : hosted hypervisors

A Type 2 hypervisor is installed on an existing OS. It's a hosted hypervisor, seeing as it relies on the host machine's OS to undertake certain operations like managing calls to the CPU, network resources, memory and storage. As Type 2 hypervisors are linked with the OS layer they can support a wide range of hardware.

Any problems in the OS affects the entire system as well even if the hypervisor running above the base OS is secure.

Conclusion

¤ OpenStack belongs to Type 1 architecture.

¤ VirtualBox belongs to Type 2 architecture.

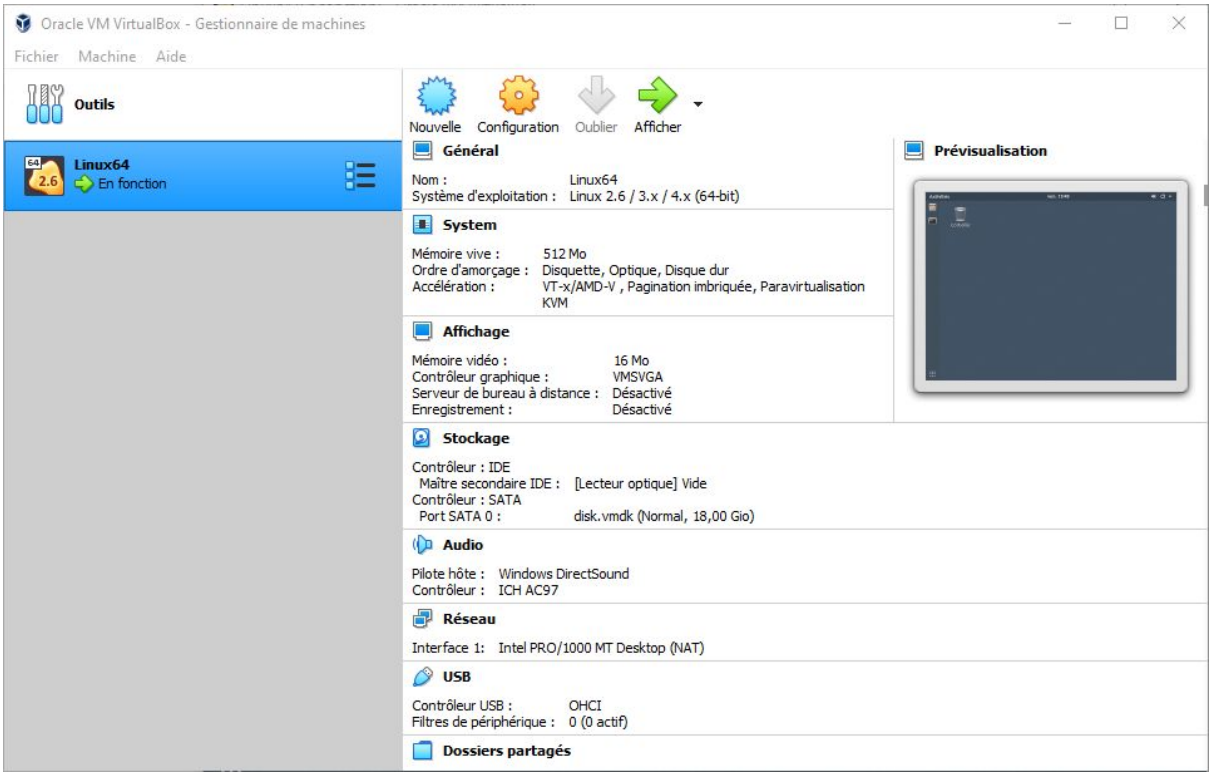
Zennaro Thomas
Licini Valentin
5ISS - B11

Practical Part

Tasks related to objectives 4 and 5

First part: Creating and configuring a VM

In this first part, we are asked to create a VM linux according to some specifications given.

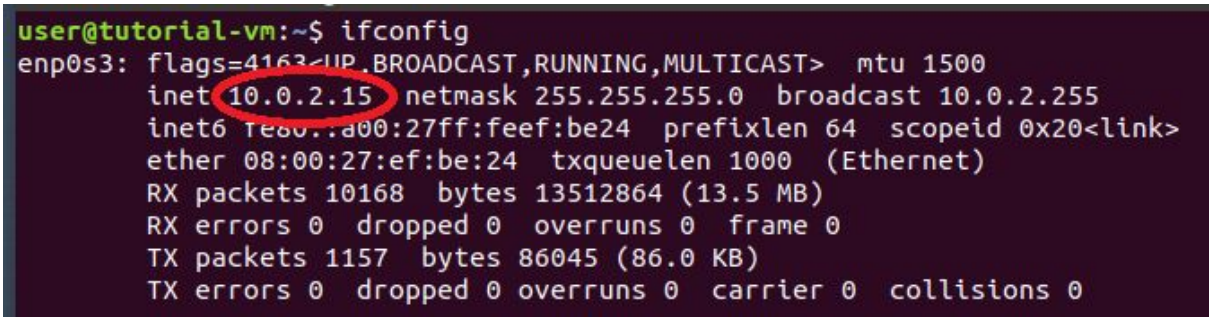


Second part: Testing the VM connectivity

The IP addresses are as follows:

- IP of the physical machine: **10.1.5.89/16**
- Virtual machine IP: **10.0.2.15/24**

As we can see from the following snapshots:



```
Suffixe DNS propre à la connexion. . . : insa-toulouse.fr
Adresse IPv6 de liaison locale. . . . .: fe80::b95c:d22a:3692:d390%4
Adresse IPv4. . . . . . . . . . . . . .: 10.1.5.89
Masque de sous-réseau. . . . . . . . . : 255.255.0.0
Passerelle par défaut. . . . . . . . . : 10.1.0.254
```

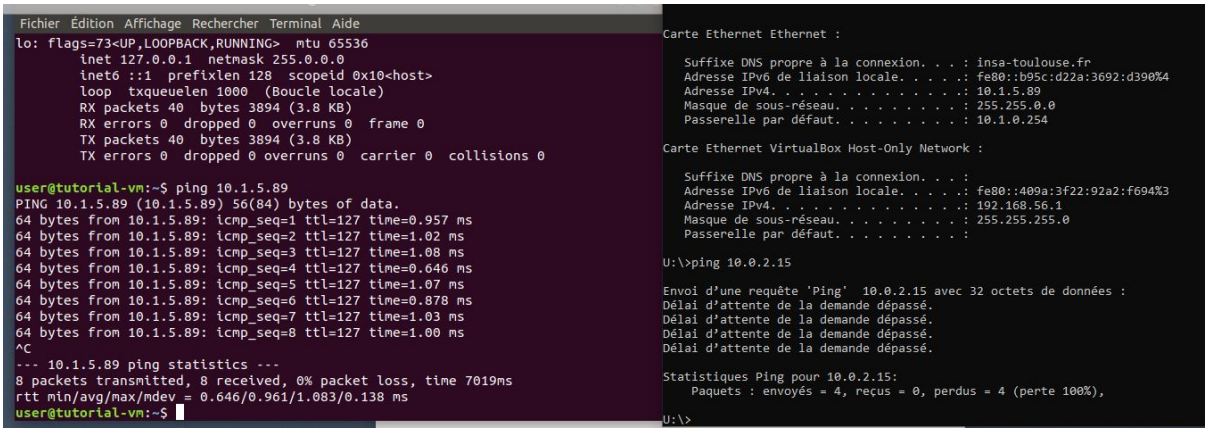
Both addresses are class A but are part of separate subnets.

We conclude that the host machine is part of the GEI subnet established by the INSA administrators who have a class A address. From this, virtualbox has created a subnet for virtual machines such as ours.

Connectivity test:

VM	-> Host	Ok
VM	-> Distant Host	Ok
Distant Host	-> VM	n/a
Host	-> VM	n/a

Examples of connectivity test using ping:



```
Fichier  Édition  Affichage  Rechercher  Terminal  Aide
lo: flags=73<UP,LOOPBACK,RUNNING>  mtu 65536
        inet 127.0.0.1  netmask 255.0.0.0
        inet6 ::1  prefixlen 128  scopeid 0x10<host>
        loop  txqueuelen 1000  (Boucle locale)
        RX packets 40  bytes 3894 (3.8 KB)
        RX errors 0  dropped 0  overruns 0  frame 0
        TX packets 40  bytes 3894 (3.8 KB)
        TX errors 0  dropped 0 overruns 0  carrier 0  collisions 0

user@tutorial-vm:~$ ping 10.1.5.89
PING 10.1.5.89 (10.1.5.89) 56(84) bytes of data.
64 bytes from 10.1.5.89: icmp_seq=1 ttl=127 time=0.957 ms
64 bytes from 10.1.5.89: icmp_seq=2 ttl=127 time=1.02 ms
64 bytes from 10.1.5.89: icmp_seq=3 ttl=127 time=1.08 ms
64 bytes from 10.1.5.89: icmp_seq=4 ttl=127 time=0.646 ms
64 bytes from 10.1.5.89: icmp_seq=5 ttl=127 time=1.07 ms
64 bytes from 10.1.5.89: icmp_seq=6 ttl=127 time=0.878 ms
64 bytes from 10.1.5.89: icmp_seq=7 ttl=127 time=1.03 ms
64 bytes from 10.1.5.89: icmp_seq=8 ttl=127 time=1.00 ms
^C
--- 10.1.5.89 ping statistics ---
8 packets transmitted, 8 received, 0% packet loss, time 7019ms
rtt min/avg/max/mdev = 0.646/0.961/1.083/0.138 ms
user@tutorial-vm:~$
```

```
Carte Ethernet Ethernet :

   Suffixe DNS propre à la connexion. . . : insa-toulouse.fr
   Adresse IPv6 de liaison locale. . . . .: fe80::b95c:d22a:3692:d390%4
   Adresse IPv4. . . . . . . . . . . . . .: 10.1.5.89
   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::409a:3f22:92a2:f694%3
   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.
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%),

U:\>
```

We infer that the subnet of the VM is connected to the network of the host machine through virtualbox which acts as a router. Since the subnet is private, it cannot be reached from outside.

Third part: Set up the "missing" connectivity

To perform the port forwarding necessary for the symmetric operation of our VM, we must configure a rule like suis, in virtualbox:
@host = PC address (insa) @guest = @VM port 22

We want to connect via PuTTY in ssh (listening on port 22) on the virtual machine:

10.1.5.89:800 ----> **10.0.2.15:22**

Fourth part: VM duplication

With an adequate “right-click” on our VM on virtualbox, we can duplicate it with the same parameters.

Fifth part: Docker containers provisioning

In this part, we will install docker in our previous VM already configured. We get the ubuntu image, then we launch it. Inside we’re already ***root*** so we don’t need to use the ***sudo*** command.

Then we retrieve the docker @IP : 172.17.0.2.

After that, we ping Google.com (which is an Internet resource) to test the connectivity with the Internet. As we see on the screen below, it works.

```
root@0c4f47389cb0:/# ping google.com
PING google.com (172.217.18.238) 56(84) bytes of data.
64 bytes from par10s10-in-f238.1e100.net (172.217.18.238): icmp_seq=1 ttl=113 time=8.08 ms
64 bytes from par10s10-in-f238.1e100.net (172.217.18.238): icmp_seq=2 ttl=113 time=7.75 ms
64 bytes from par10s10-in-f238.1e100.net (172.217.18.238): icmp_seq=3 ttl=113 time=7.80 ms
64 bytes from par10s10-in-f238.1e100.net (172.217.18.238): icmp_seq=4 ttl=113 time=7.73 ms
64 bytes from par10s10-in-f238.1e100.net (172.217.18.238): icmp_seq=5 ttl=113 time=7.59 ms
^C
```

Then, we ping the VM from Docker and it works too (see the screen below).

```
root@0c4f47389cb0:/# 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=0.054 ms
64 bytes from 10.0.2.15: icmp_seq=2 ttl=64 time=0.094 ms
64 bytes from 10.0.2.15: icmp_seq=3 ttl=64 time=0.076 ms
64 bytes from 10.0.2.15: icmp_seq=4 ttl=64 time=0.073 ms
64 bytes from 10.0.2.15: icmp_seq=5 ttl=64 time=0.077 ms
^C
```

Finally, we test the connectivity from the VM to the container and it works well as follow:

```
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=0.065 ms
64 bytes from 172.17.0.2: icmp_seq=2 ttl=64 time=0.066 ms
64 bytes from 172.17.0.2: icmp_seq=3 ttl=64 time=0.066 ms
64 bytes from 172.17.0.2: icmp_seq=4 ttl=64 time=0.063 ms
64 bytes from 172.17.0.2: icmp_seq=5 ttl=64 time=0.054 ms
^C
--- 172.17.0.2 ping statistics ---
5 packets transmitted, 5 received, 0% packet loss, time 4077ms
rtt min/avg/max/mdev = 0.054/0.062/0.066/0.011 ms
```

The docker works in bridge so it's on the same network as the host so ping works.

After that, we created a new instance namedr CT2 in which we install the nano editor.

After that, we created a snapshot of the container CT2.

```
CONTAINER ID        IMAGE               COMMAND             CREATED             STATUS              PORTS                   NAMES
b3d8eb4ce4d5        ubuntu              "/bin/bash"         3 minutes ago       Up 3 minutes        0.0.0.0:2223->22/tcp    ct2
0c4f47389cb0        ubuntu              "/bin/bash"         13 minutes ago      Up 12 minutes                               ct1
user@tutorial-vm:~$ sudo docker commit b3d8eb4ce4d5 dockerPictures:version1
invalid reference format: repository name must be lowercase
user@tutorial-vm:~$ sudo docker commit b3d8eb4ce4d5 dockerpictures:version1
sha256:a1c223730d9b8b3e3fed1001cc252657d38c2c5905895d69c343bfb975a120de
user@tutorial-vm:~$
```

We execute a new instance CT3 using the previous snapshot. The list of images is presented below.

```
user@tutorial-vm:~/Bureau$ sudo docker images
REPOSITORY          TAG                 IMAGE ID            CREATED             SIZE
dockerpictures      version2            52cce00bb03a        27 seconds ago      99.8MB
dockerpictures      version1            a1c223730d9b        26 minutes ago      99.8MB
ubuntu              latest              9140108b62dc        13 days ago         72.9MB
```

We see in the CT3 that we actually nano install. This is normal, because when taking the snapshot of CT2 all present resources are kept, so the nano editor is also present.

Expected work for objectives 6 and 7

First part : CT creation and configuration on OpenStack

The VM is attached to a private network and its @IP is 192.168.1.12. But when using ifconfig: we only observe an IPV6 address : fe80::f816:3eff:fe43:bc08/64.

Second part: Connectivity test

We noticed that a ping request from the VM to the computer host didn't work. We need to put a gateway to ensure connectivity with the public network. We decided to put all our Virtual machines in the same private network. The gateway will connect the public network to the private network.

Then we test the connectivity between our VM with the outside using ssh and it perfectly works.

Third part: Snapshot, restore and resize a VM

The resize operation consists in changing the flavor of a VM, thus allowing it to upscale or downscale according to user needs. If we resize a running VM which has a bigger configuration (RAM, SSD…) to a smaller config, we notice that the VM crashed. On the contrary, it works (smaller to bigger). OpenStack can allocate new ressources on static way but also in real time i.e. when VMs are running, that can not be the case with Virtual Box.

Making a snapshot of the VM means saving all its data to make another VM exactly the same. When we use this snapshot to create a new VM, it has all what was installed in the previous one, and its data.

Restoring the VM from the last backup erases all new data and changes made from that backup.

Zennaro Thomas
Licini Valentin
5ISS - B11

Expected work for objectives 8 and 9

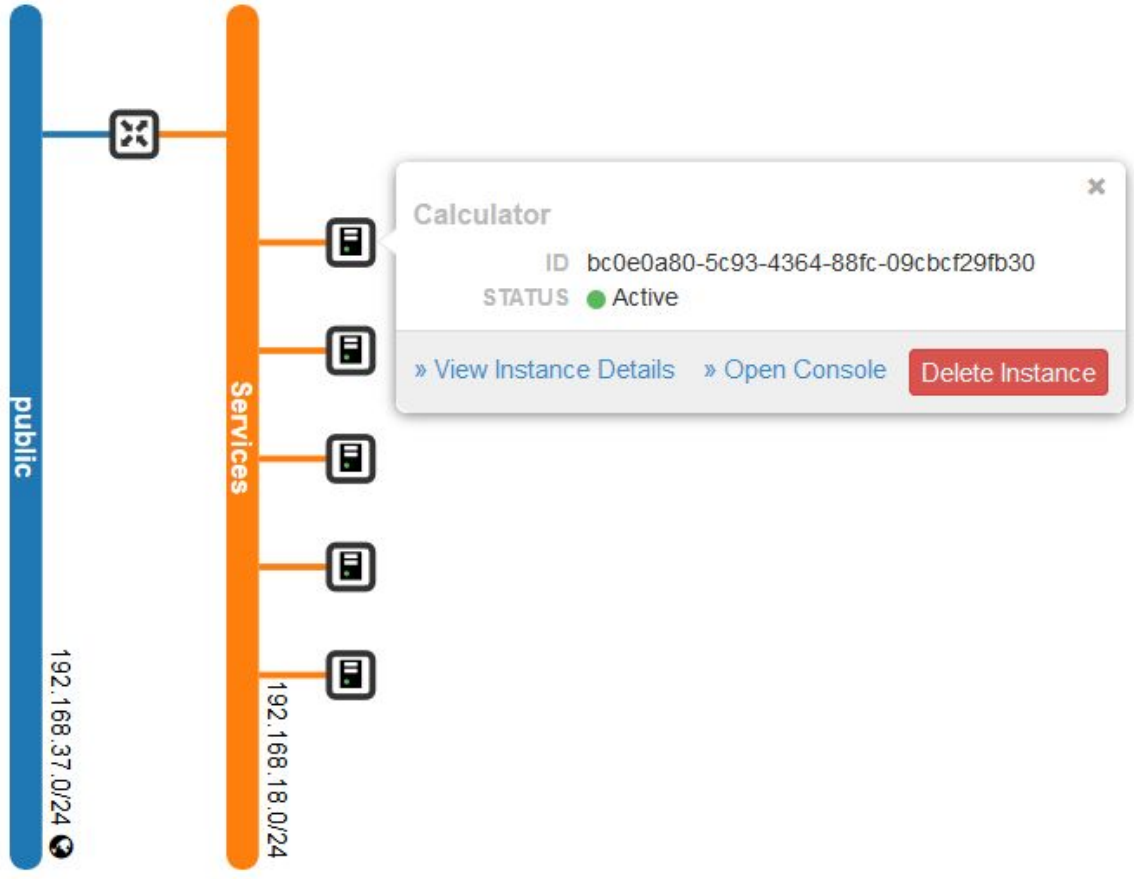
OpenStack client installation

Part three: Deploy the Calculator application on OpenStack

In order to test in an only VM all the services, we need to do the steps below:

- In the CalculatorService.js file we have to modify the IP address by “localhost”.
- In the CalculatorService.js file the port has to be chosen between 50000 and 50050
- Having 4 terminal which are listening on each port associated to each sub-service (SumService, SubService, MulService, DivService)

We create 5 VMs and each of them is going to implement one service between the 5 compute services. All of the VMs will be integrated in the same network.



For the VM which contains the calculator service, we need to put a floating IP in order to be at the interface with the external user.

We have added a new security rule in order to open the 50005 port.

As shown on the screens below, the application is running well.

```
zennaro@insa-10580:~$ curl -d "(5+6)*2" -X POST http://192.168.37.121:50005
result = 22

zennaro@insa-10580:~$
```

```
user@tutorial-vm:~/Bureau$ node CalculatorService.js
Listening on port : 50005
New request :
(5+6)*2 = 22
```

Then we used the nano editor to modify the Calculator file when it was running. After restarting it, we observed that the change is visible : the sentence “mon gros” is added on the terminal.

```
zennaro@insa-10580:~$ curl -d "(5+6)*2" -X POST http://192.168.37.121:50005
salut mon grosresult = 22
```

Part four: Orchestrate the application deployment

In this part we are asked to do a similar operation that previously. On the VM, after installing docker and the required tools (particularly the nodeJs client), we make a new container instance in which we put the 5 services. Then, we make a snapshot of this instance. We create 4 more containers based on the previous snapshot. Then, each container is assigned to a particular micro service. So, after launching the different services and changing the right @IP on the CalculatorService file, we used the CalculatorService on the first container to verify that it works. Furthermore, it also works when calling it from the VM itself.

Unfortunately, we didn’t have the time to automate the service deployment.

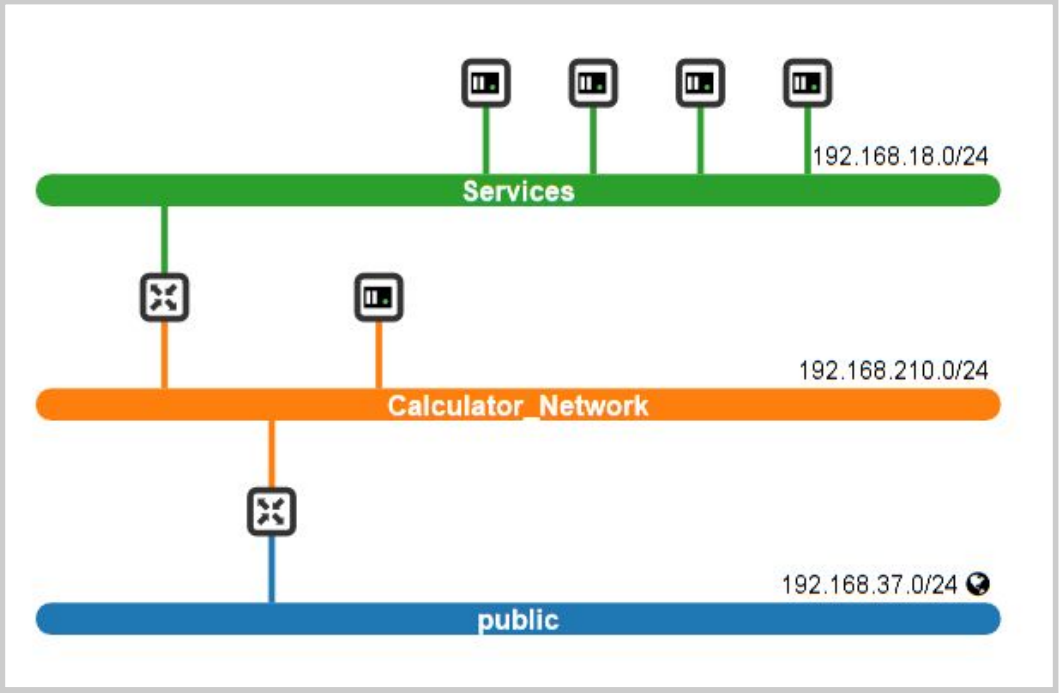
Zennaro Thomas
Licini Valentin
5ISS - B11

Expected work for objectives 10 and 11

Part two: Deployment of the target topology

We decided to continue with Ubuntu images instead of alpine-node because we could reuse the VM created in the previous part.

The built network is as follow:



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

We test the connectivity between the VM hosting the CalculatorService and the VM hosting the SumService : no response. It's normal because we have 2 private networks.
The front-end machine doesn't know where the service machines are on the internet, so it sends frames to the default route, as known as the internet router. We have to define, in its route table, which getaway to take when addressing the service machine's network. We use the following command line:
sudo route add -net 192.168.18.0/24 gw 192.168.210.1

The calculator eventually works.