Ewan Mackay - Abir Benazzouz 5ISS - 2021

Introduction to Cloud Hypervisors

Introduction

The general purpose of this practical is to enhance our familiarity with the concept and technologies of virtualization. These technologies are used in industry and familiarity with them gives a better understanding of IT environments. These environments are typically dynamic and distributed.

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

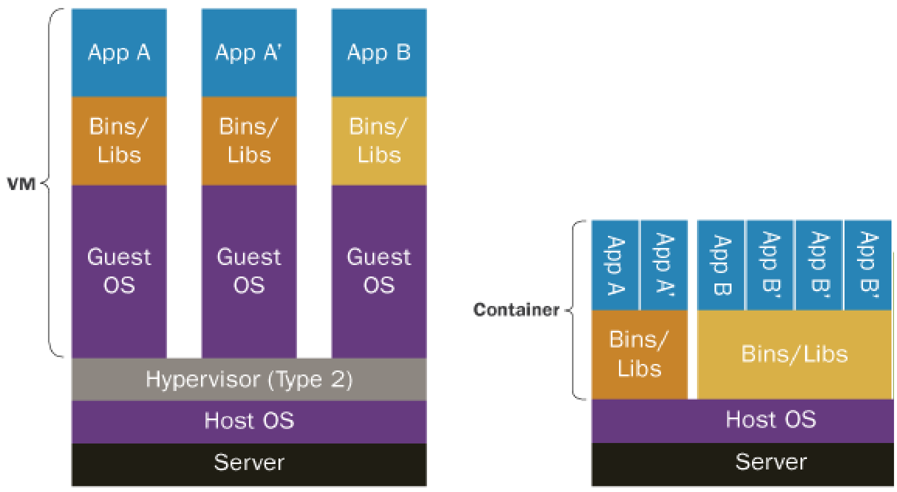


Figure 1: VM vs CT

From the illustration we can see that :
Both VMs and Containers rely on a host OS that is provided by the server layer.

- VM systems however rely on a hypervisor layer, which enables the partitioning of physical resources for each guest OS that will run independent apps. These apps and their files will therefore be run on different OSs, hosted on partitioned hardware by the hypervisor.
  In fine : Virtual machines and hypervisors abstract away hardware and enable us to run operating systems.

- Applications running in a container environment share an underlying operating system. A container not only includes the application executable, but also packs together all the necessary softwares that the application needs to run with, such as the libraries and the other dependencies.
  In fine : Containers abstract away operating systems and enable us to run applications.

A container is similar to an application, which runs as a process on top of the operating system(OS) and is isolated from each other by running in its own address space. System administrators allocate resources such as CPU, memory, network, or any combination of them, to the running containers. The resources allocated to each container can be adjusted dynamically, and the container cannot use more resources than being specified in the cgroups (the files created to specify characteristics). Namespaces provides an abstraction of the kernel resources, such as process IDs, network interfaces, host names, so that each container appears to have its own isolated resources. Containers do not require the overhead of associating an operating system within each application as is done with Virtual Machines. Containers are inherently smaller in capacity than a VM and require less start-up time, allowing far more containers to run on the same compute capacity as a single VM. This drives higher server efficiencies and, in turn, reduces server and licensing costs.

Below is a comparison of these different virtualisation techniques :

	Virtual Machine	Container
Application developer	→ Allows the execution of different OS that are not necessarily the same type as the native OS → Better security as there is more isolation between neighboring systems	→ Better solution for performance as there is no virtualization layer in a container, it incurs less performance overhead on the applications. → Better startup performance → The hardware resources, such as CPU and memory, will be returned to the operating system immediately after use. → Performs better for disk and network I/O intensive applications. → Better tooling for continuous integration support. Built in tools for DevOps.
Infrastructure administrator	→ Better security as there is more isolation between neighboring systems	→ As a lightweight solution, the size of a container is usually within tens of MB while that of a VM can take several GB. → Takes less hardware resources since it does not need to maintain an OS → resources allocated to each container can be adjusted dynamically by the system administrator

Both VMs and containers introduced negligible overhead for CPU and memory performance.

2. Similarities and differences between the existing CT types

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

Containers simplify multi-tenancy deployments by deploying multiple applications on a single host, using the kernel and the container runtime to run each container. Applications are isolated from each other and run on resources that stem from the same hardware.

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

This criteria is important to consider with regards to ease of implementation. Different container layers, like common bins and libraries, can also be shared among multiple containers which means size will change between different CT technologies.

- Tooling :

This criteria is important to consider with regards to the DevOps cycle. Maintenance of software, patches, updates, integrations within other software are all important points to consider. Effective and available tooling ensures continuation of the software's lifecycle. (e.g. API, continuous integration, or service composition)

Rank 1 Docker

Docker is the most ubiquitous containerized software solution available today. Docker uses the client-server architecture, a design in which clients request and receive service from a host, in this case, the Docker daemon. Docker has good network and file isolation. In terms of tools : Docker includes git-like capabilities for tracking successive versions of a container, shared libraries. Docker's technology is based on LXC and containers do not run an independent version of the OS kernel. Instead, all containers on a given host run under the same kernel, with other resources isolated per container. Docker can do much more than LXC.

LXC

It allows you to not only isolate applications, but even the entire OS. LXC tooling sticks close to what system administrators running bare metal servers are used to. Migrating any application from a Linux server to running on LXC containers is rather seamless. With backing from Canonical, LXC and LXD have an ecosystem tightly bound to the rest of the open source Linux community. the tools you already use on Linux will work when your applications run on LXC containers. For managing your LXC containers, the LXD hypervisor provides a clean REST API that you can use. LXD is implemented in Go, to ensure high performance and networking concurrency, with excellent integration with OpenStack and other Linux server systems.

CoreOS rkt

rkt is a more secure container technology, designed to alleviate many of the flaws inherent in Docker's container model. Docker and CoreOS will ostensibly be less concerned about dueling container standards and more focused on building a comprehensive, interoperable suite of tools for managing the entire container ecosystem. rkt clearly competes with Docker, but the two company's offerings are likely to be recommended as complementary technologies.

rkt, like all containers do, allows you to isolate your software from the environment. Some notable features about rkt are its customizable isolation and security features.rkt offers customizable isolation, which offers you a high degree of flexibility in selecting the right level of isolation. rkt can run Docker images.

In Linux, containers are just a special type of process, so securing containers is the same as securing any running process.

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

There are two main hypervisor types, referred to as “Type 1” (or “bare metal”) and “Type 2” (or “hosted”). A **type 1 hypervisor** acts like a lightweight operating system and runs directly on the host’s hardware and has a tight integration with the host kernel, while a **type 2 hypervisor** runs as a software layer on an operating system, like other computer programs. With direct access to the underlying hardware and no other software, Type 1 hypervisors are regarded as the most efficient, secure (since an attack on a guest VM is logically isolated to that VM and can't spread to others running on the same hardware) and best-performing hypervisors available. The presence of an underlying OS with Type 2 hypervisors introduces unavoidable latency; all of the hypervisors' activities and the work of every VM has to pass through the host OS. They come at a lower cost than Type 1 hypervisors and make an ideal test platform.

By this definition, VirtualBox is a Type 2 hypervisor. That is to say that it is virtualization host software that runs as an application on an established operating system. Openstack on the other hand is a software suite that enables you to set up your own Cloud environment, and corresponds more to an IaaS, with the individual hypervisors created are type 1.

PRACTICAL PART

OS (Windows):

```
U:\>ipconfig

Configuration IP de Windows


Carte Ethernet VirtualBox Host-Only Network :

   Suffixe DNS propre à la connexion. . . :
   Adresse IPv6 de liaison locale. . . . .: fe80::1834:5b3f:125b:b518%4
   Adresse IPv4. . . . . . . . . . . . . .: 192.168.56.1
   Masque de sous-réseau. . . . . . . . . : 255.255.255.0
   Passerelle par défaut. . . . . . . . . :

Carte Ethernet Ethernet :

   Suffixe DNS propre à la connexion. . . : insa-toulouse.fr
   Adresse IPv6 de liaison locale. . . . .: fe80::a9fc:1bcc:4bd7:b6ac%6
   Adresse IPv4. . . . . . . . . . . . . .: 10.1.5.71
   Masque de sous-réseau. . . . . . . . . : 255.255.0.0
   Passerelle par défaut. . . . . . . . . : 10.1.0.254

U:\>
```

Virtual NAT router address

Local physical machine address

VM :

```
user@tutorial-vm:~$ ifconfig
enp0s3: flags=4163<UP,BROADCAST,RUNNING,MULTICAST>  mtu 1500
        inet 10.0.2.15  netmask 255.255.255.0  broadcast 10.0.2.255
        inet6 fe80::a00:27ff:feae:d3d4  prefixlen 64  scopeid 0x20<link>
        ether 08:00:27:ae:d3:d4  txqueuelen 1000  (Ethernet)
        RX packets 10406  bytes 15565413 (15.5 MB)
        RX errors 0  dropped 0  overruns 0  frame 0
        TX packets 2314  bytes 153140 (153.1 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
        inet6 ::1  prefixlen 128  scopeid 0x10<host>
        loop  txqueuelen 1000  (Boucle locale)
        RX packets 36  bytes 3294 (3.2 KB)
        RX errors 0  dropped 0  overruns 0  frame 0
        TX packets 36  bytes 3294 (3.2 KB)
        TX errors 0  dropped 0 overruns 0  carrier 0  collisions 0

user@tutorial-vm:~$
```

Random machine address

Regarding the connectivity from the VM to the outside, we observe that we can ping from the VM toward the virtual card on which it is hosted, and also toward the host's physical network card. It is thus ok in a way (VM to the outside) but not the other way around.

Neither the NAT nor the Host can ping the VMs as it would make no sense for the NAT to ping all the VMs (since they have the same IP address, the NAT cannot make the difference between them).

The connectivity from our neighbour's host to our hosted VM is thus even less possible.

Third part :

By applying port forwarding, we have managed using Putty to create an SSH connection from our host to our VM, that is now associated with a specific port on the host machine. Thus we have proven the capability of communication between the host and a specific VM.
SSH connection using Putty:

```
user@tutorial-vm: ~

login as: user
user@10.1.5.71's password:
Welcome to Ubuntu 18.04.3 LTS (GNU/Linux 4.15.0-65-generic x86_64)

 * Documentation:  https://help.ubuntu.com
 * Management:     https://landscape.canonical.com
 * Support:        https://ubuntu.com/advantage

  System information as of Mon Oct  4 15:04:56 CEST 2021

  System load:  0.01              Processes:             175
  Usage of /:   21.3% of 17.59GB   Users logged in:       1
  Memory usage: 75%               IP address for enp0s3: 10.0.2.15
  Swap usage:   45%

 * Super-optimized for small spaces - read how we shrank the memory
   footprint of MicroK8s to make it the smallest full K8s around.

   https://ubuntu.com/blog/microk8s-memory-optimisation

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

Fifth part: Docker containers provisioning

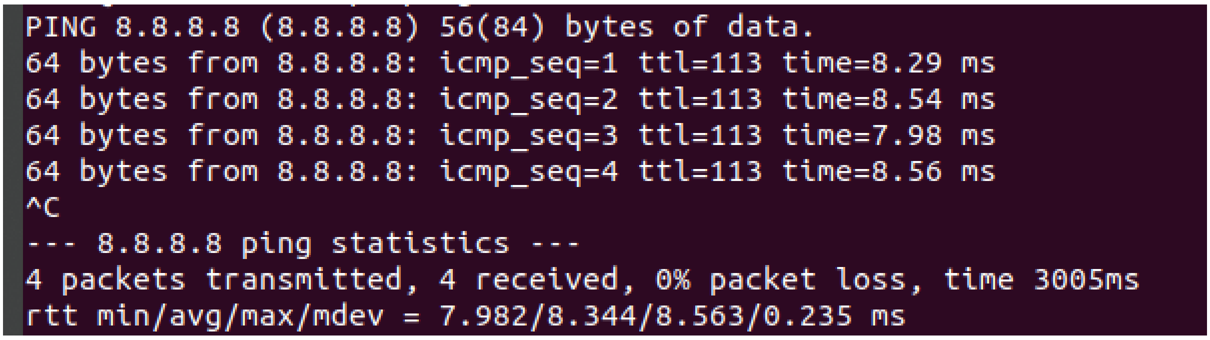
The docker has the IP address featured below :

```
root@70013eff131e:/# 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 2955  bytes 19748787 (19.7 MB)
        RX errors 0  dropped 0  overruns 0  frame 0
        TX packets 2037  bytes 115039 (115.0 KB)
        TX errors 0  dropped 0 overruns 0  carrier 0  collisions 0

lo: flags=73<UP,LOOPBACK,RUNNING>  mtu 65536
        inet 127.0.0.1  netmask 255.0.0.0
        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
```

Pinging the VM from the Docker works:

```
root@70013eff131e:/# 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=0.040 ms
64 bytes from 10.0.2.15: icmp_seq=2 ttl=64 time=0.040 ms
64 bytes from 10.0.2.15: icmp_seq=3 ttl=64 time=0.057 ms
64 bytes from 10.0.2.15: icmp_seq=4 ttl=64 time=0.086 ms
^C
```

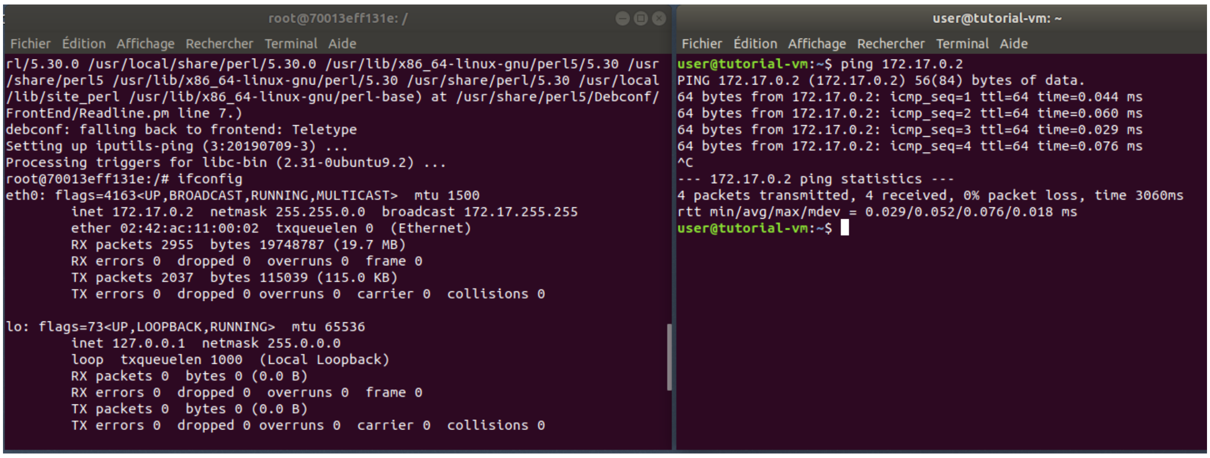
Pinging an internet resource from the Docker works:



```
PING 8.8.8.8 (8.8.8.8) 56(84) bytes of data.
64 bytes from 8.8.8.8: icmp_seq=1 ttl=113 time=8.29 ms
64 bytes from 8.8.8.8: icmp_seq=2 ttl=113 time=8.54 ms
64 bytes from 8.8.8.8: icmp_seq=3 ttl=113 time=7.98 ms
64 bytes from 8.8.8.8: icmp_seq=4 ttl=113 time=8.56 ms
^C
--- 8.8.8.8 ping statistics ---
4 packets transmitted, 4 received, 0% packet loss, time 3005ms
rtt min/avg/max/mdev = 7.982/8.344/8.563/0.235 ms
```

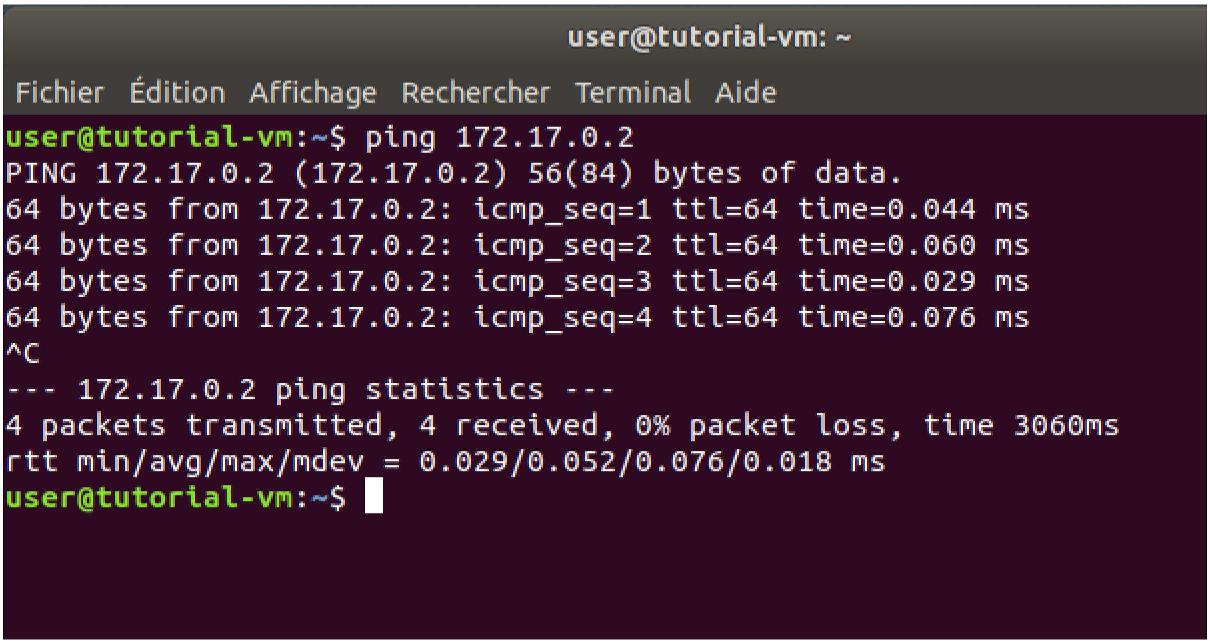
(8.8.8.8 is Google)

Pinging the docker from the VM works as well :



```
root@70013eff131e: /
Fichier  Édition  Affichage  Rechercher  Terminal  Aide
rl/5.30.0 /usr/local/share/perl/5.30.0 /usr/lib/x86_64-linux-gnu/perl5/5.30 /usr
/share/perl5 /usr/lib/x86_64-linux-gnu/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 (2.31-0ubuntu9.2) ...
root@70013eff131e:/# 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 2955  bytes 19748787 (19.7 MB)
        RX errors 0  dropped 0  overruns 0  frame 0
        TX packets 2037  bytes 115039 (115.0 KB)
        TX errors 0  dropped 0 overruns 0  carrier 0  collisions 0

lo: flags=73<UP,LOOPBACK,RUNNING>  mtu 65536
        inet 127.0.0.1  netmask 255.0.0.0
        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
```

```
user@tutorial-vm: ~
Fichier  Édition  Affichage  Rechercher  Terminal  Aide
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=0.044 ms
64 bytes from 172.17.0.2: icmp_seq=2 ttl=64 time=0.060 ms
64 bytes from 172.17.0.2: icmp_seq=3 ttl=64 time=0.029 ms
64 bytes from 172.17.0.2: icmp_seq=4 ttl=64 time=0.076 ms
^C
--- 172.17.0.2 ping statistics ---
4 packets transmitted, 4 received, 0% packet loss, time 3060ms
rtt min/avg/max/mdev = 0.029/0.052/0.076/0.018 ms
user@tutorial-vm:~$
```



```
user@tutorial-vm: ~
Fichier  Édition  Affichage  Rechercher  Terminal  Aide
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=0.044 ms
64 bytes from 172.17.0.2: icmp_seq=2 ttl=64 time=0.060 ms
64 bytes from 172.17.0.2: icmp_seq=3 ttl=64 time=0.029 ms
64 bytes from 172.17.0.2: icmp_seq=4 ttl=64 time=0.076 ms
^C
--- 172.17.0.2 ping statistics ---
4 packets transmitted, 4 received, 0% packet loss, time 3060ms
rtt min/avg/max/mdev = 0.029/0.052/0.076/0.018 ms
user@tutorial-vm:~$
```

Getting CT2's ID :

```
user@tutorial-vm:~$ sudo docker ps
[sudo] Mot de passe de user :
CONTAINER ID        IMAGE               COMMAND             CREATED
STATUS              PORTS                  NAMES
1012f81d07c6        ubuntu              "bash"              2 minutes ago
Up 2 minutes        0.0.0.0:2223->22/tcp   ct2
70013eff131e        ubuntu              "bash"              17 minutes ago
Up 17 minutes                              ct1
```

Making a snapshot of CT2 :

```
user@tutorial-vm:~$ sudo docker commit 1012f81d07c6 tp2/snap:tag1
sha256:89627e2783d3940a54c18ccb7b5ad09bbc52d9de618ab87ab9b320b3838ee797
user@tutorial-vm:~$ sudo docker images
REPOSITORY          TAG                 IMAGE ID            CREATED
SIZE
tp2/snap            tag1                89627e2783d3        6 seconds ago
105MB
ubuntu              latest              597ce1600cf4        3 days ago
72.8MB
```

The REP and Tag (tp2/snap is the repo and tag1 is the tag) arguments were chosen randomly.
cf commit command documentation : [docker commit | Docker Documentation]

Stopping the CT2 container :

```
user@tutorial-vm:~$ sudo docker rm 1012f81d07c6
Error response from daemon: You cannot remove a running container 1012f81d07c603
eb48306a9d897e56f58e8b99e212740e7954effe3cf80d0815. Stop the container before at
tempting removal or force remove
user@tutorial-vm:~$ sudo docker stop 1012f81d07c6
1012f81d07c6
user@tutorial-vm:~$ sudo docker rm 1012f81d07c6
1012f81d07c6
```

(use the stop command before)

```
user@tutorial-vm:~$ sudo docker ps
CONTAINER ID        IMAGE               COMMAND             CREATED
STATUS              PORTS               NAMES
70013eff131e        ubuntu              "bash"              29 minutes ago
Up 29 minutes                           ct1
```

Listing the available Docker images in the VM :

```
user@tutorial-vm:~$ sudo docker images
REPOSITORY          TAG                 IMAGE ID            CREATED
SIZE
tp2/snap            tag1                89627e2783d3        5 minutes ago
105MB
ubuntu              latest              597ce1600cf4        3 days ago
72.8MB
```

We notice that the CT2 image is still there even though we stopped the container.

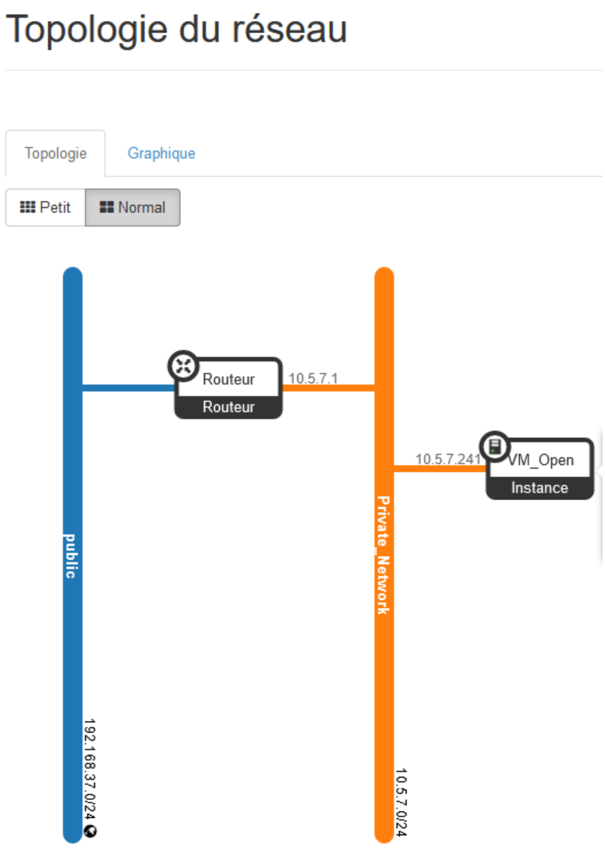
Executing new instance CT3 from the snapshot previously created :

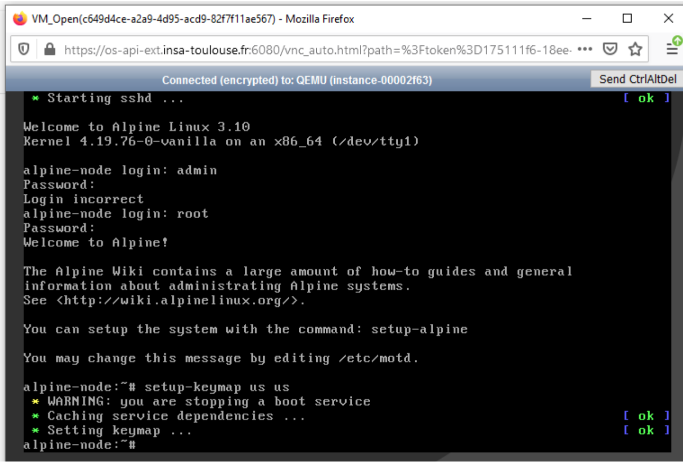
```
user@tutorial-vm:~$ sudo docker run --name ct3 -it tp2/snap:tag1
root@5eb8e9040848:/# nano
```

We notice that nano is installed on CT3. This is normal as CT3 inherited its environment from CT2, with its installed programs.

Snapshots are docker images that are saved temporarily. If the docker engine is closed, this save is lost. “Recipes” allow saving in memory in a persistent manner.

2 Expected work for objectives 6 and 7

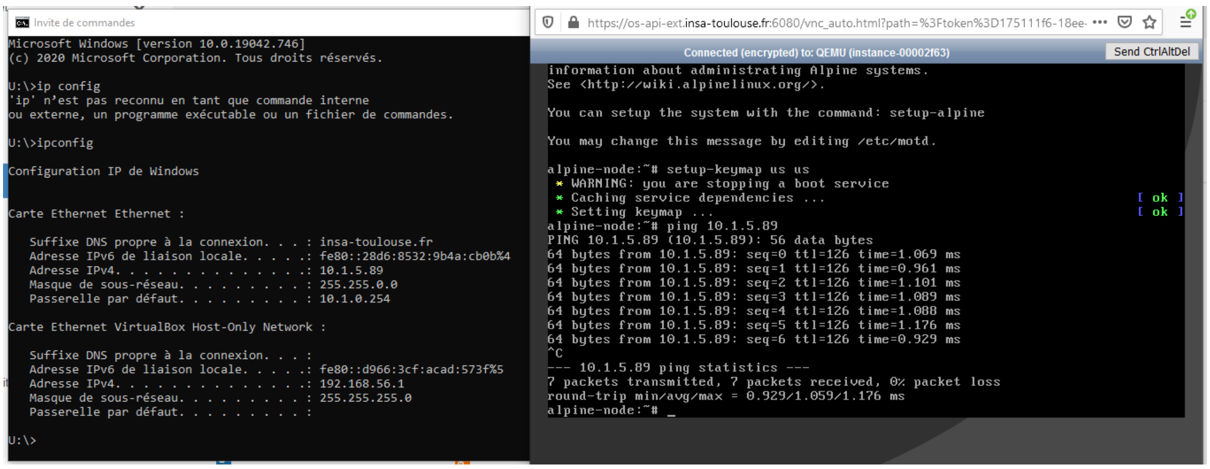




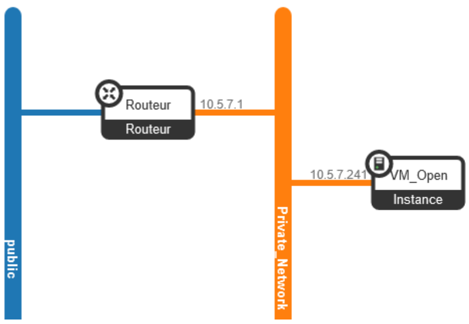
We can see that the VM has been addressed with the IP address 10.5.7.241. This seems to be a random (within acceptable range dictated by 10.5.7.0/24), free address that was given by the hypervisor.

☐	Instance Name	Image Name	IP Address
☐	VM_Open	alpine-node	10.5.7.241

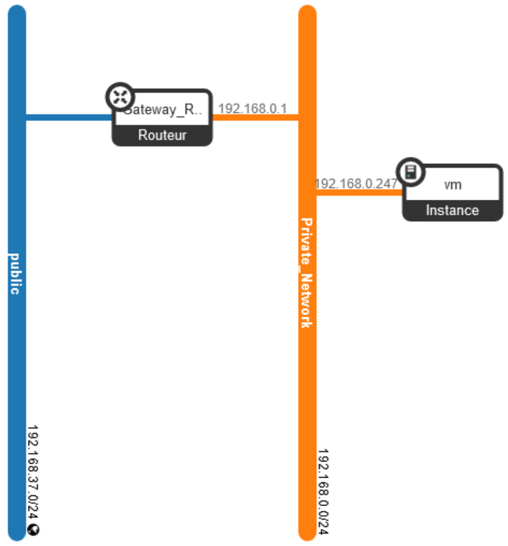
Ping works from VM to DeskTop



Ping doesn't work from DeskTop to VM



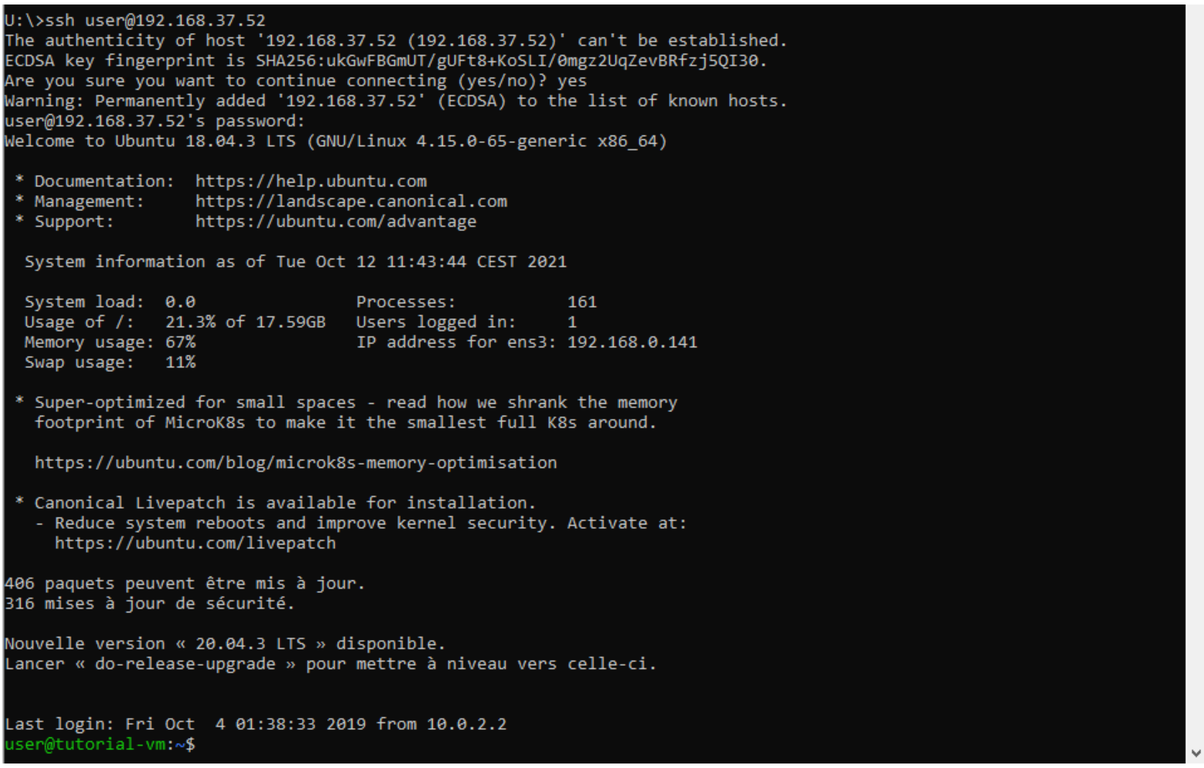
Problem : VM IP is not accessible from the INSA network. Issue with communicating with the router.
Solution : associate a floating IP address to the VM from a reserved address list for OpenStack to be able to establish bilateral communication



IP Address	Description	DNS Name	DNS Domain	Mapped Fixed IP Address	Pool	Status	Actions
192.168.37.52	VM IP			vm 192.168.0.247	public	Active	Dissocier

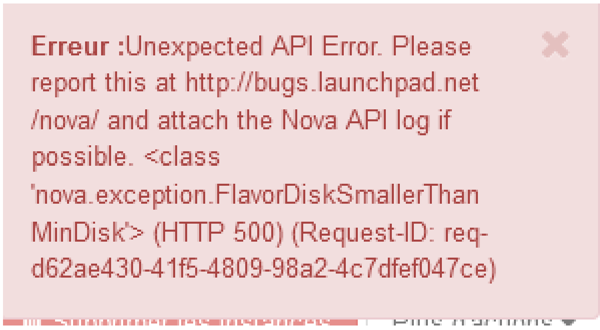
```
U:\>ping 192.168.37.52

Envoi d'une requête 'Ping'  192.168.37.52 avec 32 octets de données :
Réponse de 192.168.37.52 : octets=32 temps=4 ms TTL=62
Réponse de 192.168.37.52 : octets=32 temps=1 ms TTL=62
Réponse de 192.168.37.52 : octets=32 temps=1 ms TTL=62
Réponse de 192.168.37.52 : octets=32 temps=1 ms TTL=62

Statistiques Ping pour 192.168.37.52:
    Paquets : envoyés = 4, reçus = 4, perdus = 0 (perte 0%),
Durée approximative des boucles en millisecondes :
    Minimum = 1ms, Maximum = 4ms, Moyenne = 1ms
```

SSH Connection:



```
U:\>ssh user@192.168.37.52
The authenticity of host '192.168.37.52 (192.168.37.52)' can't be established.
ECDSA key fingerprint is SHA256:ukGwFBGmUT/gUFt8+KoSLI/0mgz2UqZevBRfzj5QI30.
Are you sure you want to continue connecting (yes/no)? yes
Warning: Permanently added '192.168.37.52' (ECDSA) to the list of known hosts.
user@192.168.37.52's password:
Welcome to Ubuntu 18.04.3 LTS (GNU/Linux 4.15.0-65-generic x86_64)

 * Documentation:  https://help.ubuntu.com
 * Management:     https://landscape.canonical.com
 * Support:        https://ubuntu.com/advantage

  System information as of Tue Oct 12 11:43:44 CEST 2021

  System load:  0.0                Processes:           161
  Usage of /:   21.3% of 17.59GB   Users logged in:     1
  Memory usage: 67%                IP address for ens3: 192.168.0.141
  Swap usage:   11%

 * Super-optimized for small spaces - read how we shrank the memory
   footprint of MicroK8s to make it the smallest full K8s around.

   https://ubuntu.com/blog/microk8s-memory-optimisation

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

406 paquets peuvent être mis à jour.
316 mises à jour de sécurité.

Nouvelle version « 20.04.3 LTS » disponible.
Lancer « do-release-upgrade » pour mettre à niveau vers celle-ci.


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

Third part : Resizing VM while running



It doesn’t work.

Expected work for objectives 6 and 7

To continue working, we need to have root access. As this is not allowed we switch to a linux based VM. After installing openstack, we can launch commands.

```
user@tutorial-vm:~$ 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>):
=========================================
address scope create                    network trunk create
address scope delete                    network trunk delete
address scope list                      network trunk list
address scope set                       network trunk set
address scope show                      network trunk show
aggregate add host                      network trunk unset
aggregate create                        network unset
aggregate delete                        object create
aggregate list                          object delete
aggregate remove host                   object list
aggregate set                           object save
```

```
(openstack) project list --help
Missing value auth-url required for auth plugin password
```

We can't see all projects as we don't have the rights. We are using the python client here, not the web client.

Part 2

After installing cURL, npm, nodejs :

we download different services with wget :

```
user@tutorial-vm:~$ wget http://homepages.laas.fr/smedjiah/tmp/SubService.js
URL transformed to HTTPS due to an HSTS policy
--2021-10-25 15:37:40--  https://homepages.laas.fr/smedjiah/tmp/SubService.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|:443… connecté.
requête HTTP transmise, en attente de la réponse… 200 OK
Taille : 746 [application/javascript]
Enregistre : «SubService.js»

SubService.js                                        100%[========================================

2021-10-25 15:37:40 (142 MB/s) - «SubService.js» enregistré [746/746]

user@tutorial-vm:~$ wget http://homepages.laas.fr/smedjiah/tmp/DivService.js
URL transformed to HTTPS due to an HSTS policy
--2021-10-25 15:37:45--  https://homepages.laas.fr/smedjiah/tmp/DivService.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|:443… connecté.
requête HTTP transmise, en attente de la réponse… 200 OK
Taille : 746 [application/javascript]
Enregistre : «DivService.js»

DivService.js                                        100%[========================================

2021-10-25 15:37:45 (195 MB/s) - «DivService.js» enregistré [746/746]
```

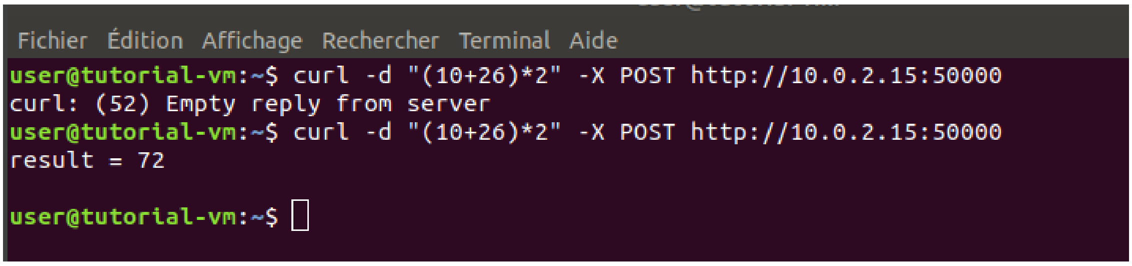
We first attempt an operation :

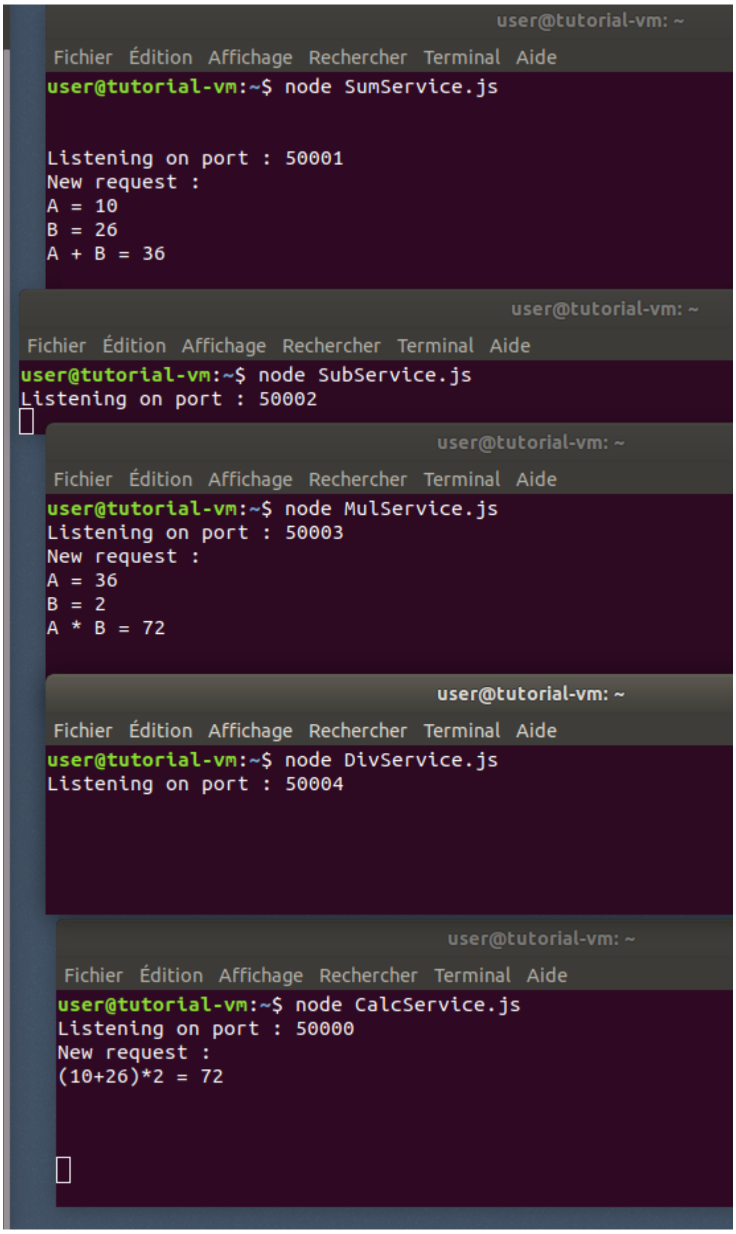
```
user@tutorial-vm:~$ node CalculatorService.js
module.js:549
    throw err;
    ^

Error: Cannot find module 'sync-request'
    at Function.Module._resolveFilename (module.js:547:15)
    at Function.Module._load (module.js:474:25)
    at Module.require (module.js:596:17)
    at require (internal/module.js:11:18)
    at Object.<anonymous> (/home/user/CalculatorService.js:2:15)
    at Module._compile (module.js:652:30)
    at Object.Module._extensions..js (module.js:663:10)
    at Module.load (module.js:565:32)
    at tryModuleLoad (module.js:505:12)
    at Function.Module._load (module.js:497:3)
user@tutorial-vm:~$ npm install sync-request
/home/user
└── sync-request@6.1.0
  ├─┬ http-response-object@3.0.2
  │ └── @types/node@10.17.60
  ├─┬ sync-rpc@1.3.6
  │ └── get-port@3.2.0
  └─┬ then-request@6.0.2
    ├── @types/concat-stream@1.6.1
    ├── @types/form-data@0.0.33
    ├── @types/node@8.10.66
    ├── @types/qs@6.9.7
    ├── caseless@0.12.0
    ├─┬ concat-stream@1.6.2
```

However some components are missing. After installation, we attempt an execution

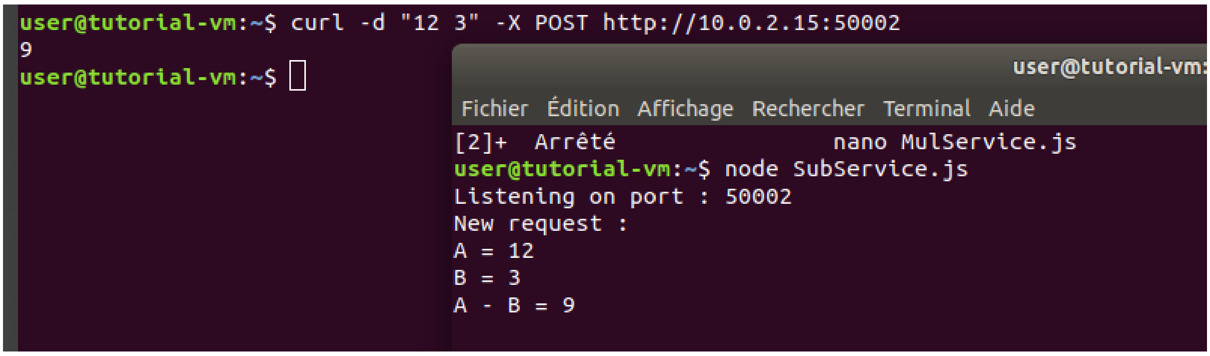
```
user@tutorial-vm:~$ ifconfig
enp0s3: flags=4163<UP,BROADCAST,RUNNING,MULTICAST>  mtu 1500
        inet 10.0.2.15  netmask 255.255.255.0  broadcast 10.0.2.255
        inet6 fe80::a00:27ff:fe9e:418d  prefixlen 64  scopeid 0x20<link>
        ether 08:00:27:9e:41:8d  txqueuelen 1000  (Ethernet)
        RX packets 35276  bytes 52261204 (52.2 MB)
        RX errors 0  dropped 0  overruns 0  frame 0
        TX packets 2587  bytes 256011 (256.0 KB)
        TX errors 0  dropped 0 overruns 0  carrier 0  collisions 0

lo: flags=73<UP,LOOPBACK,RUNNING>  mtu 65536
        inet 127.0.0.1  netmask 255.0.0.0
        inet6 ::1  prefixlen 128  scopeid 0x10<host>
        loop  txqueuelen 1000  (Boucle locale)
        RX packets 256  bytes 29502 (29.5 KB)
        RX errors 0  dropped 0  overruns 0  frame 0
        TX packets 256  bytes 29502 (29.5 KB)
        TX errors 0  dropped 0 overruns 0  carrier 0  collisions 0

user@tutorial-vm:~$ curl -d "(10+26)*2" -X POST http://10.0.2.15:50000
curl: (7) Failed to connect to 10.0.2.15 port 50000: Connexion refusée
```

But we need to change the specific ip addresses within the different services. Then launch each service in their own terminal window. Finally we can create our request that will exploit each micro-service :





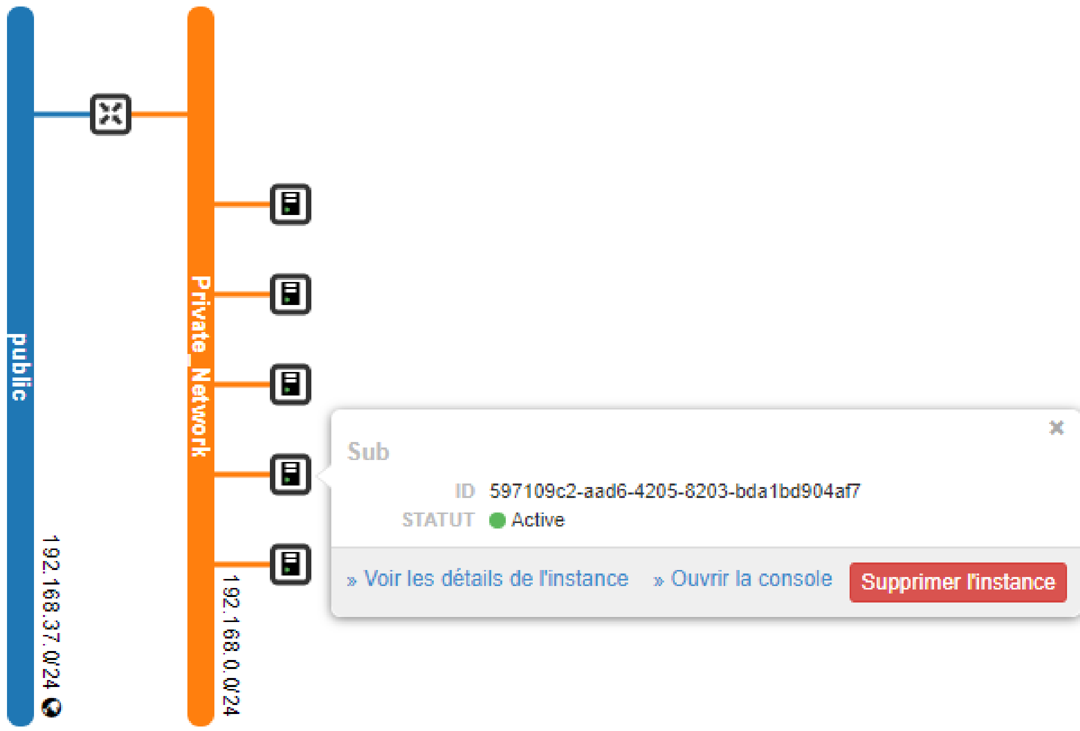
We can also use microservices completely independently from each other :



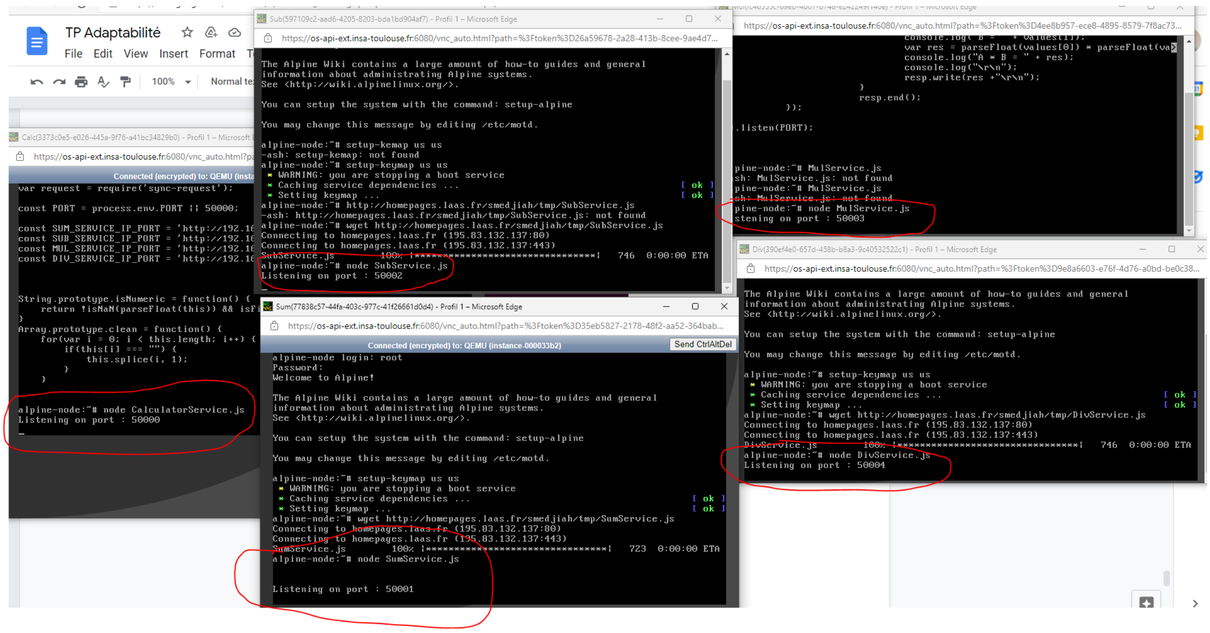
These micro services are written in javascript.

Part three:

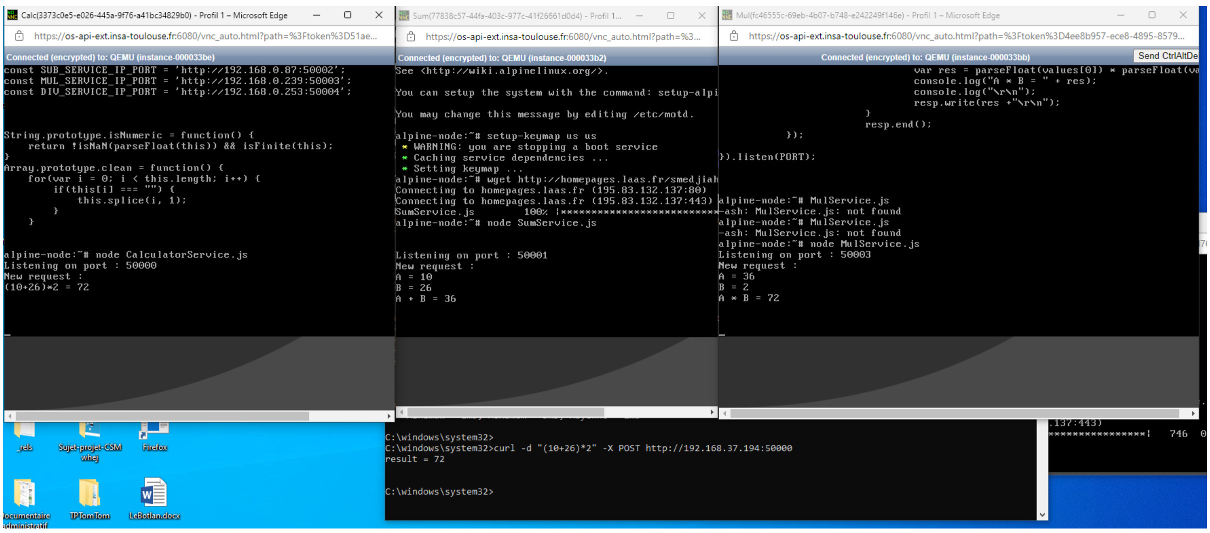
On utilise l'interface web d'openstack. On crée 5 VMs tournant chacun un µservice.



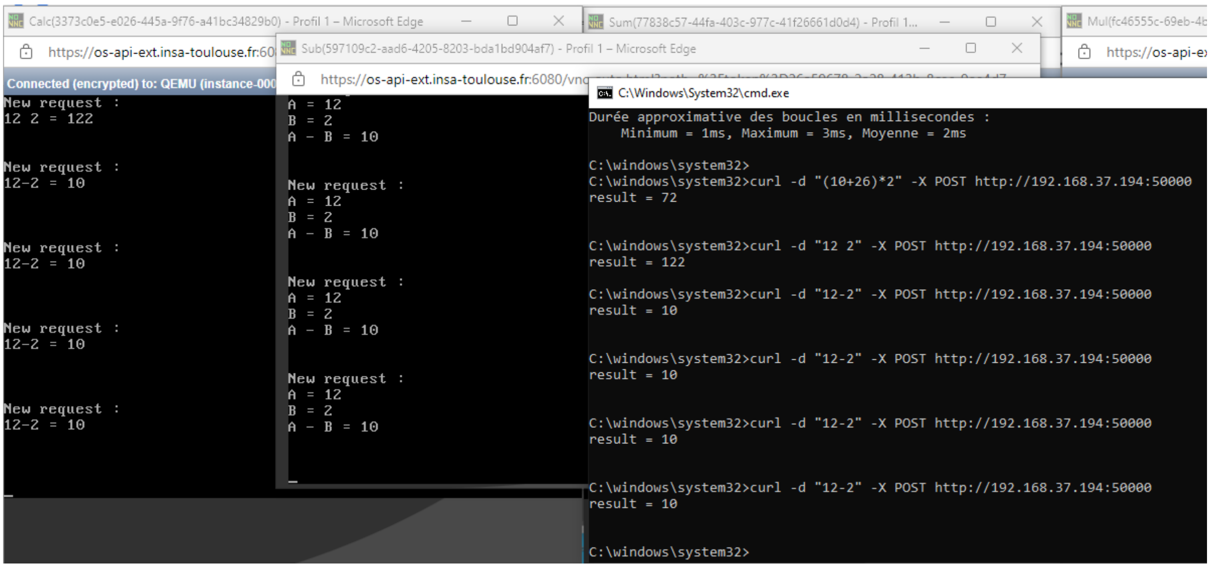
We launch all the µS on the individual and dedicated VMs.



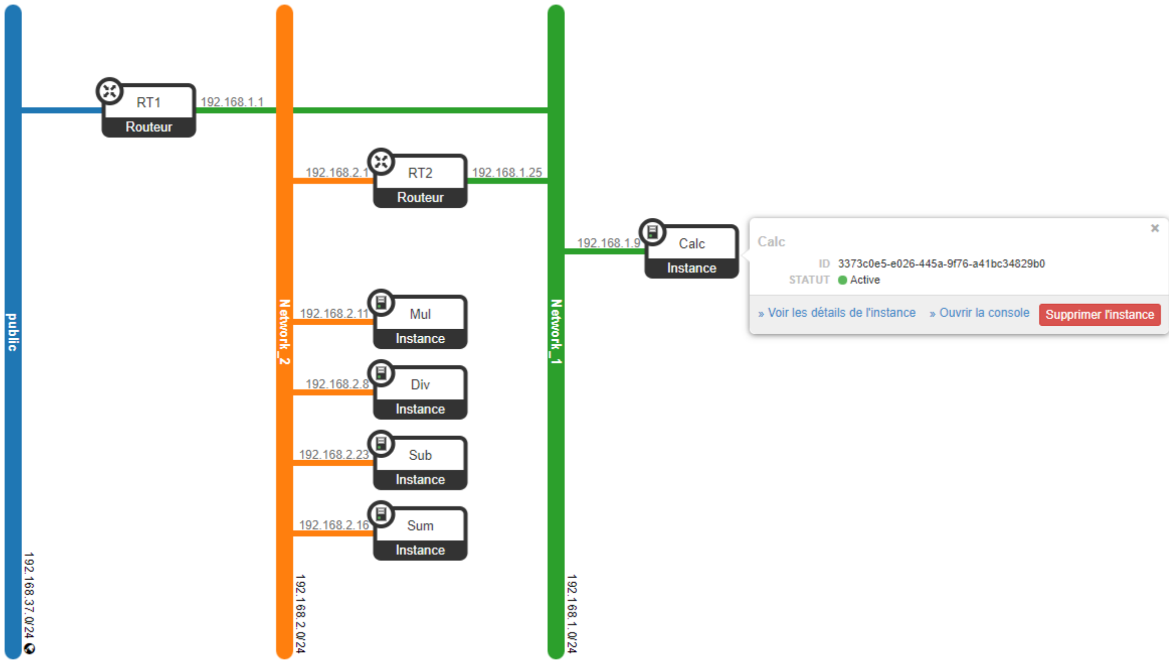
We can see that the µServices work as planned by sending the message to the floating ip associated with the calc service, who's code has been modified to address each service on their respective VMs:

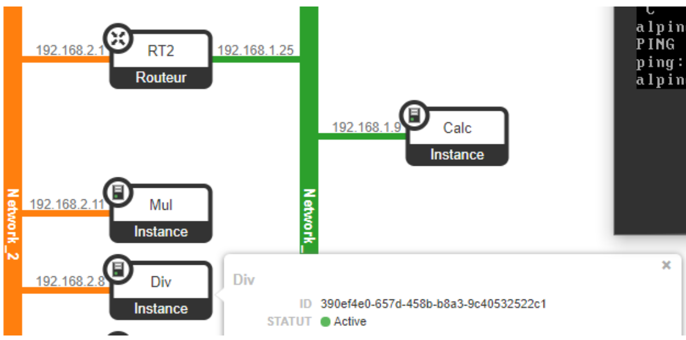


other example :



The service composition works!





The connectivity is impossible between calc and µservice.