Lab 1 : Introduction to Cloud Hypervisors

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

Application developer's point of view :

	Virtual Machine	Containers
Virtualization cost, taking into consideration memory size and CPU	- Requires higher memory size because each VM includes its own operating system. - More expensive to virtualize because it emulates an entire machine on the hardware level	- Requires less memory size because the CT doesn't duplicate the OS and the application is packaged to function on all types of environments. - Lightweight and cheaper to host
Usage of CPU, memory and network for a given application	Higher CPU overhead due to the emulation of hardware and presence of multiple OS instances ⇒ more control over the allocated resources	Lower CPU overhead because they run as lightweight processes on the host OS.
Security for the application (access right, resources sharing)	Strong isolation between applications because each VM runs its own complete operation system ⇒ more secure	Containers share the same kernel and libraries as the host system, which means that they are more vulnerable to attacks and exploits. If one container is compromised, it can affect the others and the host system. ⇒ security flaws
Performances (response time)	Higher response time because for each operation you have to regenerate the environment.	Lower response time, startup in milliseconds.
Tooling for the continuous integration support	No widespread development tools, at least by default.	Usually comes with development kits.
Flexibility, dynamicity	Hard to change the disk size	Can easily change allocated

	and resources allocated to a virtual machine, and requires more work to transfer programs.	resources for a given container or application, and requires minimal code to transfer and upload work.

Table 2 :

	Virtual Machine	Container
Developer point of view	Although virtual machines are bulkier to host and offer less flexibility, they prove valuable in situations where emulating entire machines is necessary, such as low-level programming requirements. Virtual machines are also beneficial when precise control over network infrastructure is essential. However, for most cases, using a virtual machine is excessive. The high virtualization cost, coupled with extended boot and response times, makes it a less favorable option for developers.	The portability and flexibility of containers offer significant advantages to developers. They enable cost-effective and dynamic hosting for applications due to their lightweight nature. Additionally, the capability to share resources among various containers is a valuable feature for app development. The swift startup and response times are ideal for testing and deploying programs efficiently. Moreover, the development kits bundled with containers serve as clear benefits for developers.
System Administrator point of view	Virtual Machines possess several key features that system administrators highly value. Each virtual machine is completely isolated, ensuring enhanced security and independence. They come with their own operating systems, providing access to specific tools tailored to those systems. The drawback of being heavyweight is often less significant for system administrators dealing with	Containers pose significant challenges for system administrators due to their inherent security risks, limited control over hardware, and network configurations. These factors make containers less than ideal tools for system administration tasks.

	extensive systems, as they typically have abundant resources at their disposal to manage these virtualized environments effectively.	

Overall, it’s better for the Developer to use the CT and for the Admin to use the VM.

2) Similarities and differences between the existing CT types :

Let's elaborate on the criteria we will use to compare the two container technologies (CT) under study:

1. Application Isolation and Resources (Multi-tenancy):
- Definition: Multitenancy refers to the server's structure hosting containers, where a single instance of software (the host OS) serves multiple tenants (the containers).
- Description: Application isolation and resources gauge how separated an application within a container is from other containers on the same server. It also examines how the resources, whether shared or dedicated, are allocated among these containers.

2. Containerization Level (e.g., Operating System, Application):
- Definition: Containerization involves encapsulating software and its dependencies, enabling it to run on any hardware.
- Description: The containerization level pertains to the extent of components bundled with the encapsulated software. It also considers the level of abstraction the container maintains concerning the host machine. This criterion helps understand the depth of encapsulation and abstraction achieved by each container technology.

3. Tooling (e.g., API, Continuous Integration, Service Composition):
- Definition: Tooling refers to the tools provided with the container service to assist users, which can include development kits, migration tools, custom settings, and more.
- Description: This criterion evaluates the availability and effectiveness of tools associated with each container technology. It encompasses aspects such as the presence of APIs, support for continuous integration, and capabilities related to service composition. Evaluating the tooling provides insights into the user-friendliness and additional features that each container technology offers.

	Linux LXC	**Docker**
Application isolation and resources	LxC provides OS level isolation and allows for fine grained resource allocation. It is suitable for multi-tenancy (when several different cloud customers are accessing the same computing resources) ⇒ Focuses on isolating entire operating systems	Docker provides application level isolation through its use of container images (contains everything needed to run a containerized application) ⇒It focuses on isolating individual applications within containers
Containerization level	LXC is virtualizing on OS level.	Docker is based on LXC and adds higher level functionalities. Docker provide andengine to supervise the containerization, and virtualizes on the application level.
Tooling	LxC has a CLI interface for managing containers. While it has API support, it may not be as developer-friendly as Docker.	Docker is well-known for its user-friendly CLI and API. It has strong CI/CD support with Docker Compose and Kubernetes integration. Docker Compose is useful for service composition.

3) Similarities and differences between Type 1 & Type 2 of hypervisor's 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, while a type 2 hypervisor runs as a software layer on an operating system, like other computer programs.
Type 1 hypervisors are typically favored for enterprise environments and production systems where performance and security are crucial. Type 2 hypervisors are more commonly used in desktop or development environments for convenience and ease of use.

	Type 1 (OpenStack)	Type 2 (VirtualBox)
Architecture	Runs directly on the bare-metal hardware of the physical host machine. They do not require an underlying operating system.	Runs on top of an existing operating system (the host OS). They are essentially applications or processes running within a conventional operating system.
Performance	They typically offer better performance because they have direct access to hardware resources without the overhead of an operating system layer in between.	They tend to have more overhead compared to Type 1 hypervisors because they rely on the host OS to manage hardware resources.
Use cases	They are commonly used in enterprise data centers and cloud environments where performance, scalability, and resource isolation are critical.	They are often used for desktop or developer environments where performance is not a primary concern.

OpenStack is a cloud infrastructure platform designed for large-scale cloud environments, while VirtualBox is a desktop virtualization solution for running virtual machines on a personal computer or workstation.

Practical part
(objectives 4 to 7)

1. Tasks related to objectives 4 and 5

First Part : Creating and configuring a VM

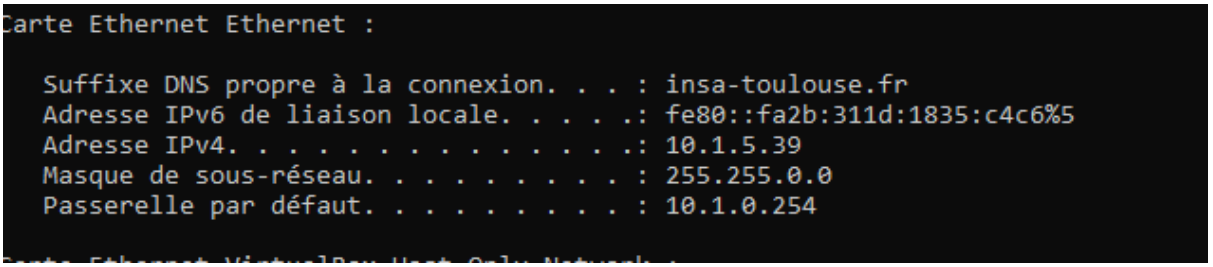
We created and configured the VM on VirtualBox after downloading the virtual hard drive.

Second Part : Testing the VM connectivity

In this part, we will be testing the connectivity of the VM by using the command ping.
But first, we need the IP addresses of both machines : the local and the VM.

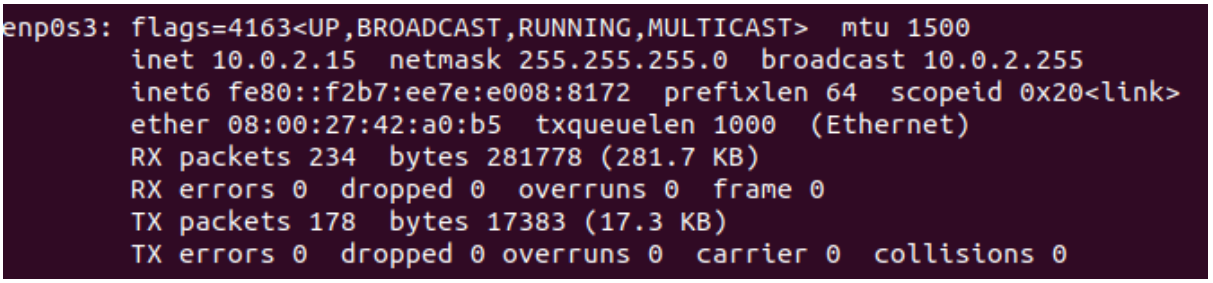
To do so we will use the **ipconfig** command :

Local Machine :



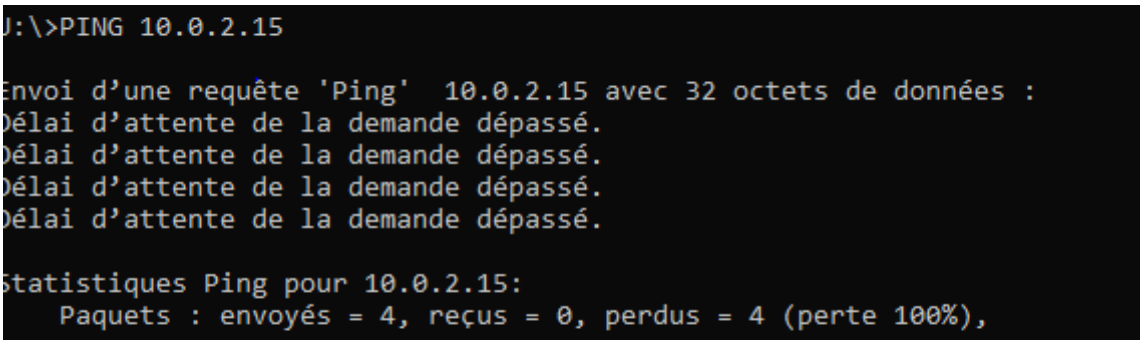
⇒ Host IP address : 10.1.5.39

VM :



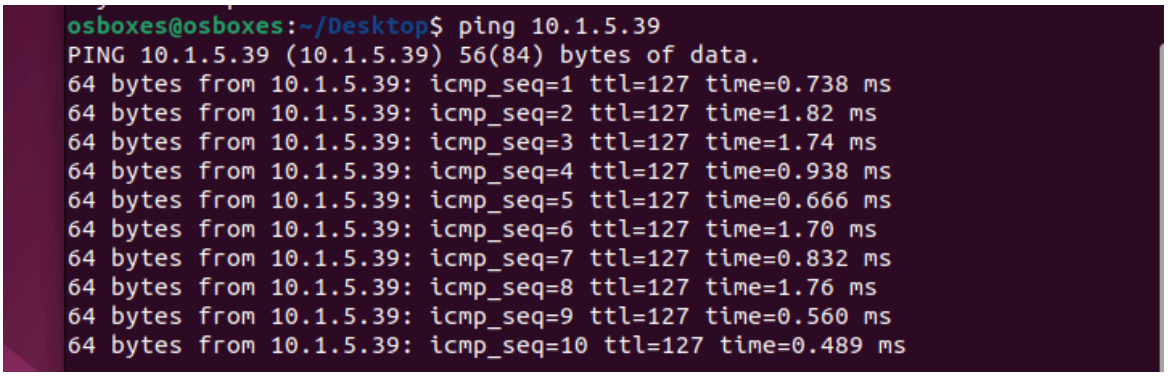
⇒ VM IP address : 10.0.2.15

Ping from local to the VM :



The VM can't be reached because it doesn't have a reachable IP address due to the different network domains.
⇒ expected from NAT, as the machine connects to a private network managed by the hypervisor

Ping from the VM to the local machine :



⇒ possible thanks to the NAT

Third part: Set up the "missing" connectivity

In order to enable external network access to the virtual machine, bypassing the NAT protocol, a port forwarding rule was established. This involved directing incoming requests on a designated port of the host to a specified port on the virtual machine, specifically port 22, which is the SSH port. Consequently, sending a request to the host's 1234 port allows for an SSH connection to the virtual machine from an external source.

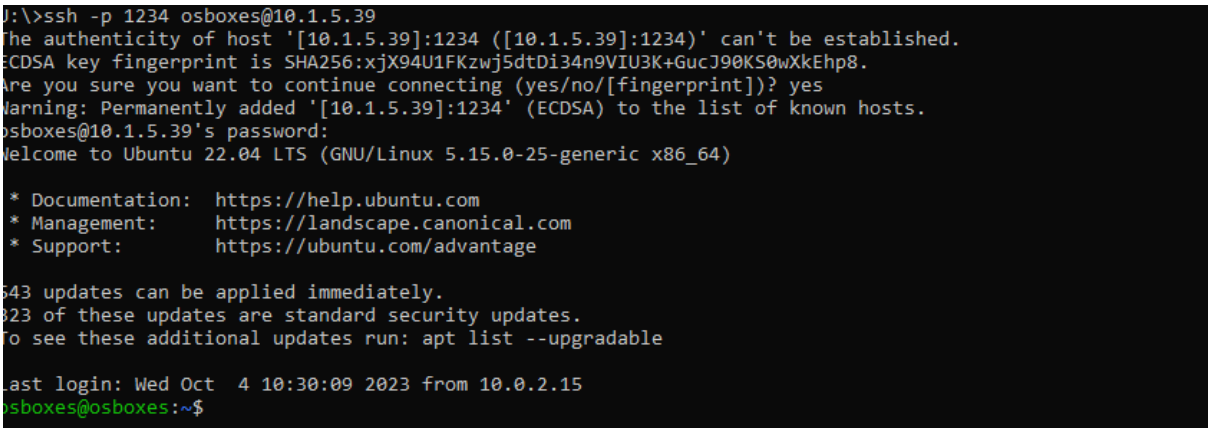
In order to succeed, we need to add a new forwarding rule to the management interface of

the network board provided by VirtualBox.

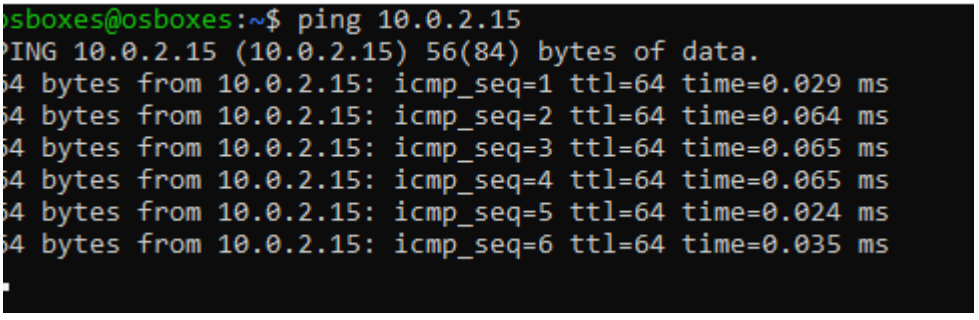
We will have something like that :

Name	Protocol	Host IP	Host Port	Guest IP	Guest Port
SSH	TCP	10.1.5.39	1234	10.0.2.15	22

Connecting via ssh client :

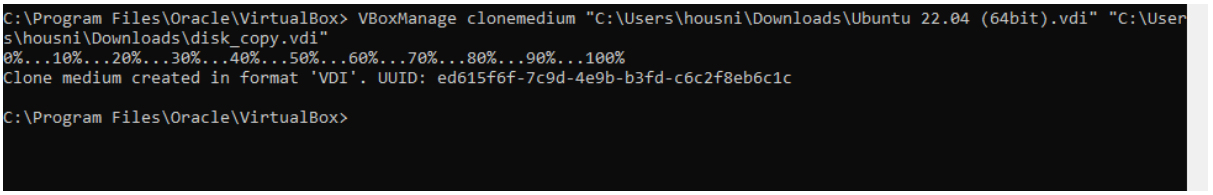


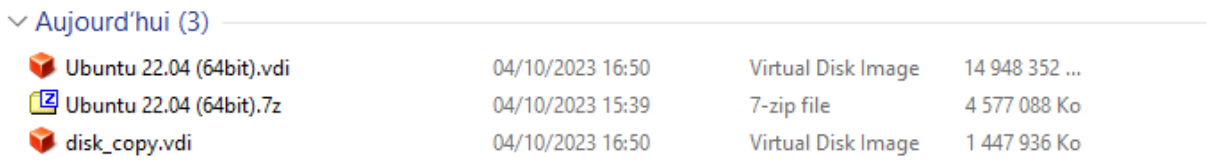
Now we succeed when trying to ping the vm from the local host :



Fourth part: VM duplication

In this part, we will clone the VM with the same disk file :





The disk was copied successfully.

Fifth part: Docker containers provisioning

In this part, we will target the Docker environment. We will first install the Docker Engine on the VM then we will use it to create and handle Docker containers.

Update of the list of packages :

Docker Engine Setup :

After setting up the environment by running these commands :

```
$ sudo apt update
```

```
$ sudo apt install apt-transport-https ca-certificates curl
software-properties-common
```

```
$ curl -fsSL https://download.docker.com/linux/ubuntu/gpg | sudo gpg
--dearmor -o /usr/share/keyrings/docker-archive-keyring.gpg
```

```
$ echo "deb [arch=$(dpkg --print-architecture)
signed-by=/usr/share/keyrings/docker-archive-keyring.gpg]
https://download.docker.com/linux/ubuntu $(lsb_release -cs) stable" | sudo
tee /etc/apt/sources.list.d/docker.list > /dev/null
```

```
$ sudo apt update
```

```
$ apt-cache policy docker-ce
```

We will install docker

```
osboxes@osboxes:~$ sudo apt install docker-ce
Reading package lists... Done
Building dependency tree... Done
Reading state information... Done
Suggested packages:
  aufs-tools cgroupfs-mount | cgroup-lite
The following packages will be upgraded:
  docker-ce
1 upgraded, 0 newly installed, 0 to remove and 521 not upgraded.
Need to get 22.6 MB of archives.
After this operation, 8,496 kB of additional disk space will be used.
Get:1 https://download.docker.com/linux/ubuntu jammy/stable amd64 docker-ce amd64 5:24.0.6-1~ubunt
ammy [22.6 MB]
Fetched 22.6 MB in 1s (19.8 MB/s)
```

Check if it's running :

```
sboxes@osboxes:~$ sudo systemctl status docker
● docker.service - Docker Application Container Engine
     Loaded: loaded (/lib/systemd/system/docker.service; enabled; vendor preset: enabled)
     Active: active (running) since Wed 2023-10-04 11:13:13 EDT; 1min 5s ago
TriggeredBy: ● docker.socket
       Docs: https://docs.docker.com
   Main PID: 10786 (dockerd)
      Tasks: 8
     Memory: 14.7M
        CPU: 244ms
     CGroup: /system.slice/docker.service
             └─10786 /usr/bin/dockerd -H fd:// --containerd=/run/containerd/containerd.sock

Oct 04 11:13:12 osboxes dockerd[10786]: time="2023-10-04T11:13:12.870887784-04:00" level=info msg="
Oct 04 11:13:12 osboxes dockerd[10786]: time="2023-10-04T11:13:12.876441712-04:00" level=info msg="
Oct 04 11:13:12 osboxes dockerd[10786]: time="2023-10-04T11:13:12.979944853-04:00" level=info msg="
Oct 04 11:13:12 osboxes dockerd[10786]: time="2023-10-04T11:13:12.988272691-04:00" level=info msg="
Oct 04 11:13:13 osboxes dockerd[10786]: time="2023-10-04T11:13:13.562781336-04:00" level=info msg="
Oct 04 11:13:13 osboxes dockerd[10786]: time="2023-10-04T11:13:13.619330379-04:00" level=info msg="
Oct 04 11:13:13 osboxes dockerd[10786]: time="2023-10-04T11:13:13.718272668-04:00" level=info msg="
Oct 04 11:13:13 osboxes dockerd[10786]: time="2023-10-04T11:13:13.718525483-04:00" level=info msg="
Oct 04 11:13:13 osboxes dockerd[10786]: time="2023-10-04T11:13:13.769726265-04:00" level=info msg="
Oct 04 11:13:13 osboxes systemd[1]: Started Docker Application Container Engine.
lines 1-22/22 (END)
            docker info
```

⇒ Docker running.

Docker Containers Provisioning

Now, we are ready to provision Docker nodes.

First, we get an image of ubuntu :

```
osboxes@osboxes:~$ sudo docker pull ubuntu
Using default tag: latest
latest: Pulling from library/ubuntu
37aaf24cf781: Pull complete
Digest: sha256:9b8dec3bf938bc80fbe758d856e96fdfab5f56c39d44b0cff351e847bb1b01ea
Status: Downloaded newer image for ubuntu:latest
docker.io/library/ubuntu:latest
```

Then, we execute an instance of the ubuntu image by running the command :

```
$ sudo docker run --name ct1 -it ubuntu
```

Finally, we install the connectivity tools :

```
root@9853187d72fa:/# apt-get -y update && apt-get -y install net-tools iputils-ping
Get:1 http://security.ubuntu.com/ubuntu jammy-security InRelease [110 kB]
```

Check the connectivity (through ping) with the newly instantiated Docker:

1. What is the Docker IP address?

```
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 3734  bytes 28116373 (28.1 MB)
        RX errors 0  dropped 0  overruns 0  frame 0
        TX packets 2079  bytes 117220 (117.2 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
```

⇒ Docker IP address : 172.17.0.2

2. Ping an Internet resource from Docker

```
root@9853187d72fa:/# ping google.com
PING google.com (142.250.200.238) 56(84) bytes of data.
64 bytes from mrs08s18-in-f14.1e100.net (142.250.200.238): icmp_seq=1 ttl=113 time=6.44 ms
64 bytes from mrs08s18-in-f14.1e100.net (142.250.200.238): icmp_seq=2 ttl=113 time=7.10 ms
64 bytes from mrs08s18-in-f14.1e100.net (142.250.200.238): icmp_seq=3 ttl=113 time=6.34 ms
64 bytes from mrs08s18-in-f14.1e100.net (142.250.200.238): icmp_seq=4 ttl=113 time=7.44 ms
64 bytes from mrs08s18-in-f14.1e100.net (142.250.200.238): icmp_seq=5 ttl=113 time=6.57 ms
64 bytes from mrs08s18-in-f14.1e100.net (142.250.200.238): icmp_seq=6 ttl=113 time=6.61 ms
```

⇒ works properly

3. Ping the VM from Docker

```
root@9853187d72fa:/# 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=0.034 ms
64 bytes from 10.0.2.15: icmp_seq=2 ttl=64 time=0.036 ms
64 bytes from 10.0.2.15: icmp_seq=3 ttl=64 time=0.034 ms
64 bytes from 10.0.2.15: icmp_seq=4 ttl=64 time=0.032 ms
64 bytes from 10.0.2.15: icmp_seq=5 ttl=64 time=0.036 ms
```

⇒ works properly

4. Ping the Docker from the VM

```
osboxes@osboxes:~$ 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=0.027 ms
64 bytes from 172.17.0.2: icmp_seq=2 ttl=64 time=0.031 ms
64 bytes from 172.17.0.2: icmp_seq=3 ttl=64 time=0.030 ms
64 bytes from 172.17.0.2: icmp_seq=4 ttl=64 time=0.031 ms
64 bytes from 172.17.0.2: icmp_seq=5 ttl=64 time=0.029 ms
64 bytes from 172.17.0.2: icmp_seq=6 ttl=64 time=0.027 ms
```

⇒ works properly

5. Elaboration on the obtained results

Anticipated outcomes include the distinction between a virtual machine (VM) that virtualizes at the hardware level and Docker, a container that operates at the OS level. The container has the capability to access its host's resources and network due to its lighter encapsulation.

Now, we will execute a new instance of the ubuntu docker and installing nano :

```
osboxes@osboxes:~$ sudo docker run --name ct12 -p 2223:22 -it ubuntu
[sudo] password for osboxes:
root@bae8da556291:/# apt-get -y update && apt install nano
Get:1 http://security.ubuntu.com/ubuntu jammy-security InRelease [110 kB]
Get:2 http://archive.ubuntu.com/ubuntu jammy InRelease [270 kB]
Get:3 http://security.ubuntu.com/ubuntu jammy-security/universe amd64 Packages [
1002 kB]
Get:4 http://archive.ubuntu.com/ubuntu jammy-updates InRelease [119 kB]
Get:5 http://archive.ubuntu.com/ubuntu jammy-backports InRelease [109 kB]
Get:6 http://archive.ubuntu.com/ubuntu jammy/main amd64 Packages [1792 kB]
Get:7 http://security.ubuntu.com/ubuntu jammy-security/restricted amd64 Packages
 [1223 kB]
Get:8 http://archive.ubuntu.com/ubuntu jammy/multiverse amd64 Packages [266 kB]
Get:9 http://security.ubuntu.com/ubuntu jammy-security/multiverse amd64 Packages
 [44.0 kB]
Get:10 http://security.ubuntu.com/ubuntu jammy-security/main amd64 Packages [107
3 kB]
```

checking the ID of the docker instances in order to make a snapshot :

```
osboxes@osboxes:~$ sudo docker ps
[sudo] password for osboxes:
CONTAINER ID   IMAGE     COMMAND       CREATED          STATUS          PORTS
                                NAMES
bae8da556291   ubuntu    "/bin/bash"   8 minutes ago    Up 8 minutes    0.0.0.0:
2223->22/tcp, :::2223->22/tcp   ct12
9853187d72fa   ubuntu    "/bin/bash"   20 minutes ago   Up 20 minutes
                                ct11
```

snapshot of CT2 :

```
osboxes@osboxes:~$ sudo docker commit bae8da556291 container2:version1
sha256:02462a18337c20f17e55ce457f1ffd40ae9cd6e9b9c89df70219c36ed0af8552
osboxes@osboxes:~$
```

Stopping and terminating CT2 :

```
osboxes@osboxes:~$ sudo docker stop bae8da556291
bae8da556291
osboxes@osboxes:~$ sudo docker rm bae8da556291
bae8da556291
```

Listing the available Docker images in the VM :

```
osboxes@osboxes:~$ sudo docker images
REPOSITORY    TAG         IMAGE ID       CREATED              SIZE
container2    version1    02462a18337c   About a minute ago   124MB
<none>        <none>      11bae9f5a8ca   3 minutes ago        124MB
```

We can see that the snapshot of CT2 is still available.

Executing a new instance (CT3) from the snapshot that we previously created from CT2 using the run command

```
osboxes@osboxes:~$ sudo docker run --name ct13 -it container2:version1
root@06622d99a4f2:/# nano --help
Usage: nano [OPTIONS] [[+LINE[,COLUMN]] FILE]...

To place the cursor on a specific line of a file, put the line number with
a '+' before the filename.  The column number can be added after a comma.
When a filename is '-', nano reads data from standard input.

 Option         Long option             Meaning
 -A             --smarthome             Enable smart home key
 -B             --backup                Save backups of existing files
 -C <dir>       --backupdir=<dir>       Directory for saving unique backup files
 -D             --boldtext              Use bold instead of reverse video text
```

We notice that nano is already installed

⇒ This is due to the utilization of shared resources on the host machine by containers. Upon installing nano in container2, the files were downloaded into the shared resources, and permissions were granted to container2 for accessing them. Upon creating the snapshot, these permissions were replicated, allowing the new container to access the files with nano properly configured.

Now, we will create a recipe by creating a dockerfile :

```
FROM ubuntu

RUN apt update -y

RUN apt install -y nano

CMD ["/bin/bash"]
```

Then, we build the image in the VM :

```
osboxes@osboxes:~$ sudo docker build -t container3:version2 -f MyDocker.dockerfi
le .
[+] Building 10.1s (7/7) FINISHED
 => [internal] load build definition from MyDocker.dockerfile                0.1s
 => => transferring dockerfile: 125B                                         0.1s
 => [internal] load .dockerignore                                            0.0s
 => => transferring context: 2B                                              0.0s
 => [internal] load metadata for docker.io/library/ubuntu:latest             0.0s
 => [1/3] FROM docker.io/library/ubuntu                                      0.0s
 => [2/3] RUN apt update -y                                                  5.9s
 => [3/3] RUN apt install -y nano                                            3.5s
 => exporting to image                                                       0.5s
 => => exporting layers                                                      0.5s
 => => writing image sha256:f9d9fe0bf3b5b4f1b35448fbeaa6478c06f61e36cf1d0    0.0s
 => => naming to docker.io/library/container3:version2                       0.0s
osboxes@osboxes:~$
```

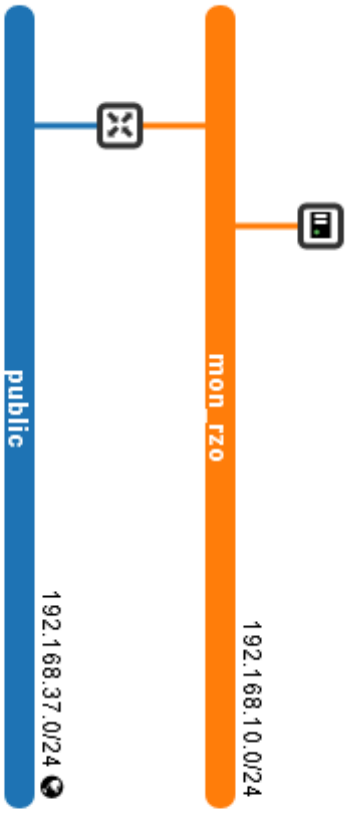
2. Expected work for objectives 6 and 7

First part: CT creation and configuration on OpenStack

First we will create an instance using the ubuntu4CLV and small2 flavor.

⇒ error, when trying to run the instance.

The virtual machine fails to initiate as the network is unable to assign it an IP address. Due to the security configurations of the public network, the new VM is not recognized as a network participant, leading to the denial of an IP address. To address this, it is necessary to configure a private network capable of accommodating our VMs. This involves incorporating two rules, namely SSH and ICMP, into the security group. Subsequently, the network topology is configured with a private network housing our VM, along with a gateway facilitating communication between the public and private networks.



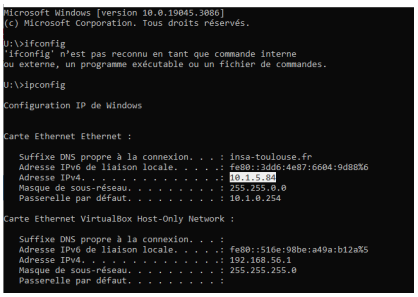
Second part: Connectivity test

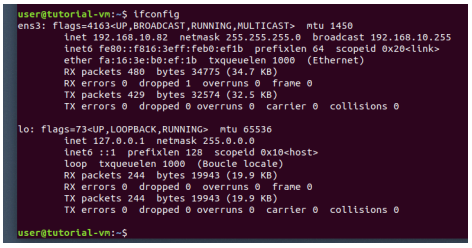
Now, we will create our own private network and customize it to accept new rules. We will also setup a gateway to allow the communication between our private network and the public network.

We can see that an IP address has been attributed to the VM, now we will test the connectivity using the command **ping**.

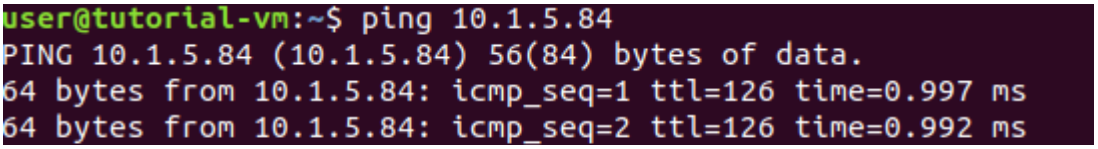
Local machine IP address : 10.1.5.84

VM IP address : 192.168.10.82/24

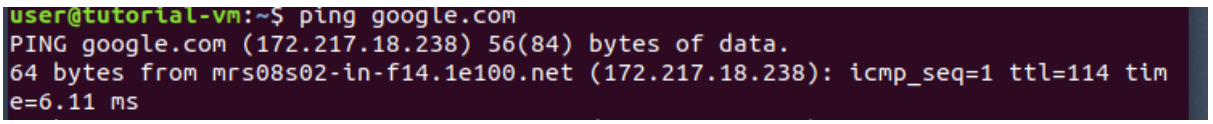




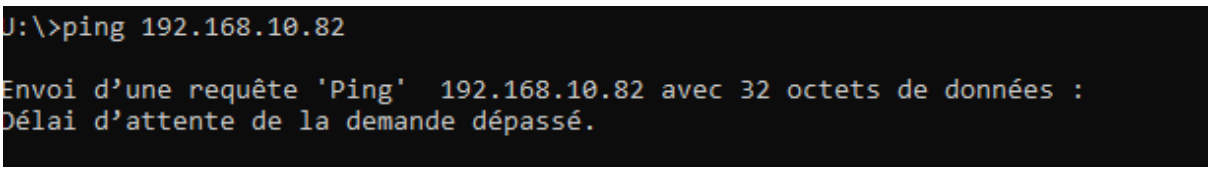
Ping from the VM to the local machine PC : OK



Ping from the VM to the internet : OK



Ping from the local machine to the VM : KO

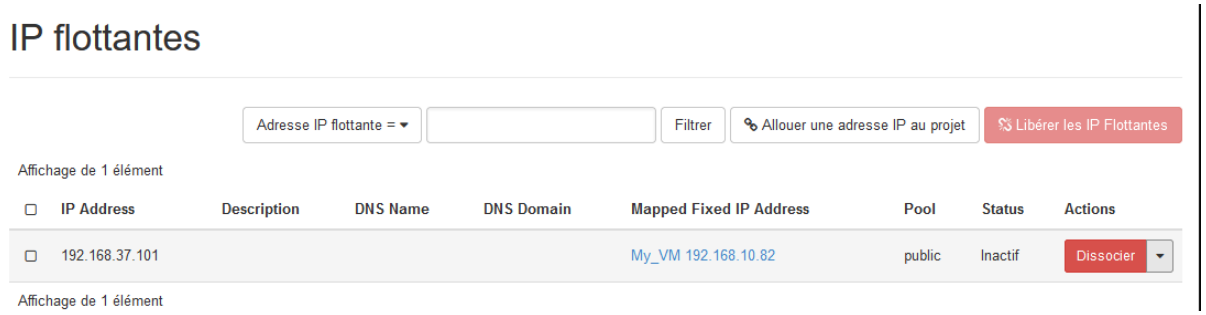


⇒ not within the same network, indicating that the unreachable address belongs to a private

networ : unable to route any packets to it.

FLOATING IP :

To reach the VM, we have to create a floating IP address and allocate it to the VM.



The gateway that we created implements a NAT like VirtualBox.

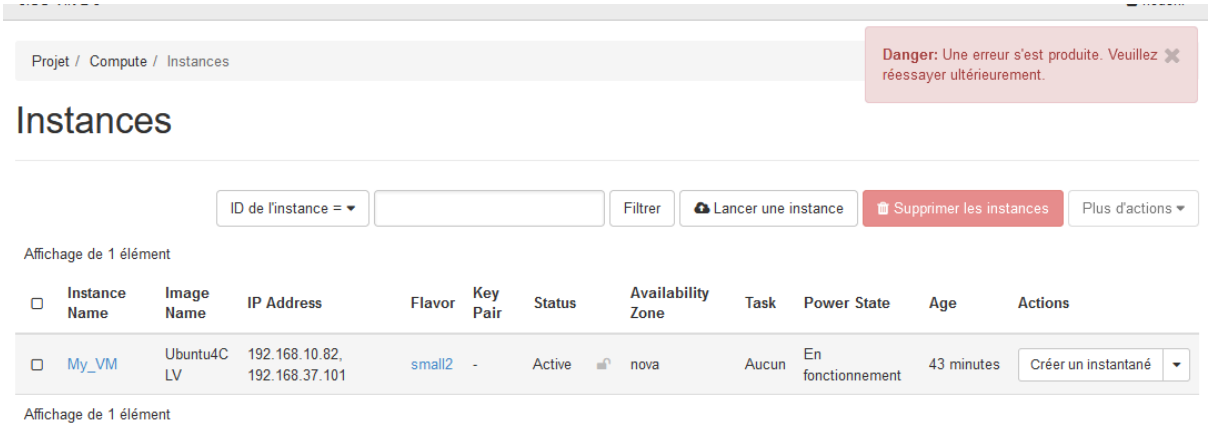
Now, we will try to ping the VM using the floating IP : OK

```
U:\>ping 192.168.37.101

Envoi d'une requête 'Ping'  192.168.37.101 avec 32 octets de données :
Réponse de 192.168.37.101 : octets=32 temps=1 ms TTL=62
Réponse de 192.168.37.101 : octets=32 temps=1 ms TTL=62
Réponse de 192.168.37.101 : octets=32 temps=1 ms TTL=62
Réponse de 192.168.37.101 : octets=32 temps=1 ms TTL=62

Statistiques Ping pour 192.168.37.101:
    Paquets : envoyés = 4, reçus = 4, perdus = 0 (perte 0%),
Durée approximative des boucles en millisecondes :
    Minimum = 1ms, Maximum = 1ms, Moyenne = 1ms
```

We will try the connection with SSH as well : OK

```
ECDSA key fingerprint is SHA256:8kfcVNQvumv3iZK/UewZP/MyQrdrj3rfTnY7I5XpYF4.
Are you sure you want to continue connecting (yes/no/[fingerprint])? yes
Warning: Permanently added '192.168.37.101' (ECDSA) to the list of known hosts.
user@192.168.37.101'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 10 13:46:13 CEST 2023

  System load:  0.0               Processes:           161
  Usage of /:   20.5% of 17.59GB  Users logged in:     1
  Memory usage: 66%               IP address for ens3: 192.168.10.82
  Swap usage:   3%


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

0 paquet peut être mis à jour.
0 mise à jour de sécurité.

Failed to connect to https://changelogs.ubuntu.com/meta-release-lts. Check your Internet connection or proxy settings


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

Third part: Snapshot, restore and resize a VM

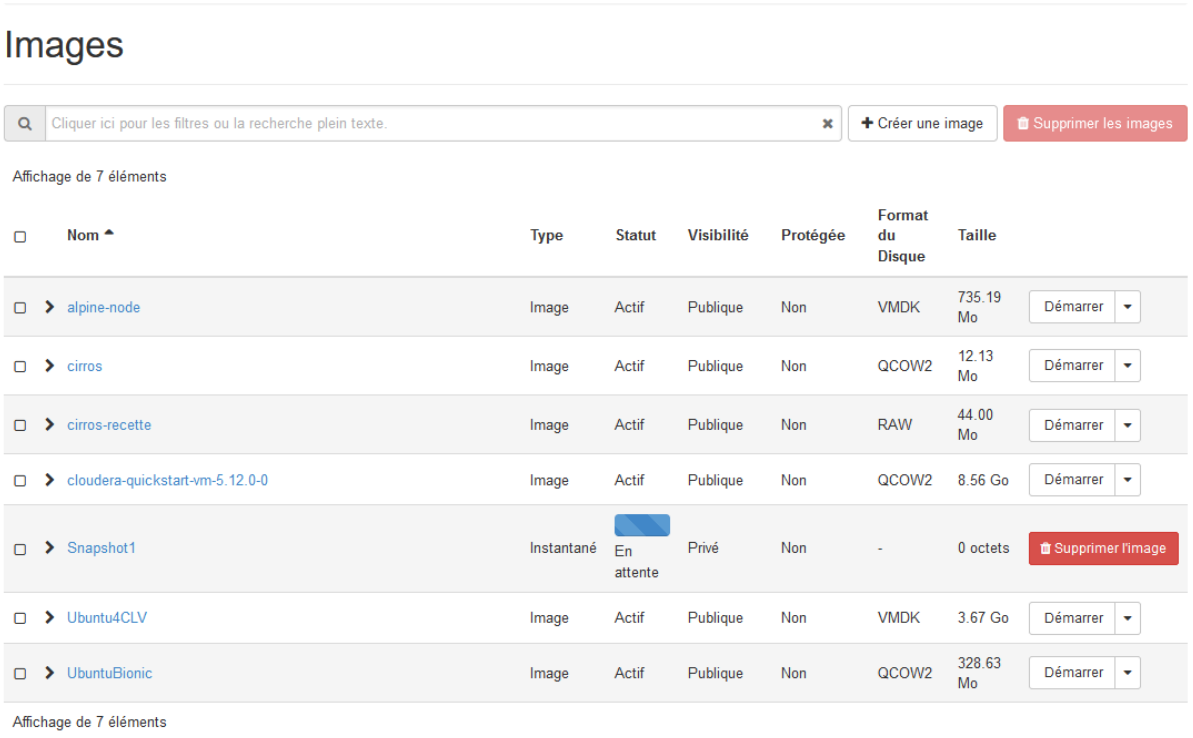


We tried resizing the VM but it didn’t work.

We tried again after shutting the VM down, still not workin.

⇒ INSA has restricted the resizing capability due to cost considerations.

- Make a snapshot of the VM. Elaborate on the differences between the included material/software within the original image and the newly created snapshot



It appears that all the machine's parameters and software are identical; however, the network settings differ slightly. When creating a VM with the snapshot, the IP address is not the same as that of the original machine. This indicates that they are treated as distinct entities by the network

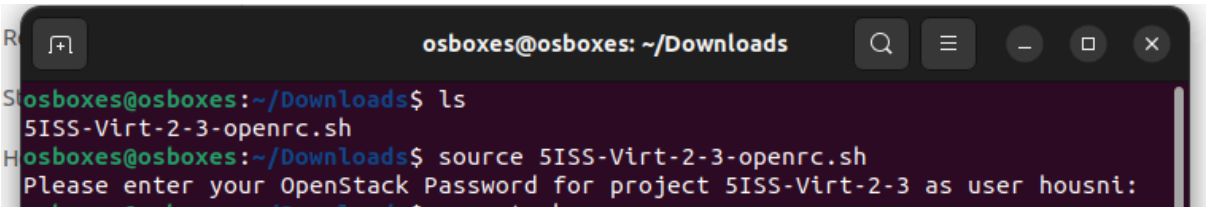
3. Expected work for objectives 8 and 9.

Part one : OpenStack client installation

First, we install the openstack client by running :

```
$ sudo apt install python3-openstackclient
```

Then, we configure the client with the appropriate variables by sourcing the RC file that we downloaded from the Openstack API Dashboard.

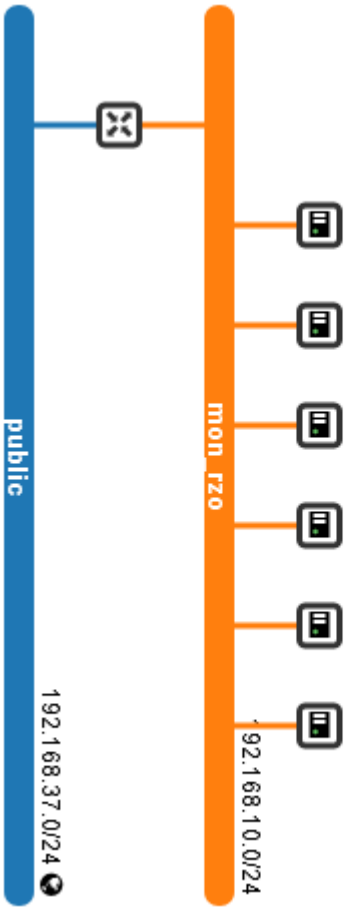


Finally, we start the client :

Part two : Web 2-tier application topology and specification

The purpose of this section is the deployment of a 2-tier Web application on OpenStack. The application is designed to execute arithmetic operations (addition, subtraction, multiplication, and division) involving integer numbers, with each operation implemented through a NodeJs microservice. Additionally, there is a fifth microservice responsible for handling the application endpoint. This service actively listens for and receives incoming HTTP-based requests, specifying the arithmetic operations to be computed by the application. The results of these operations are then displayed on both the front-end component and the client.

We deploy the new network topology on OpenStack :



The first thing to do is to download the different services using the command **wget**.

Then preparing the environment by running these commands :

```
$ sudo apt install nodejs
```

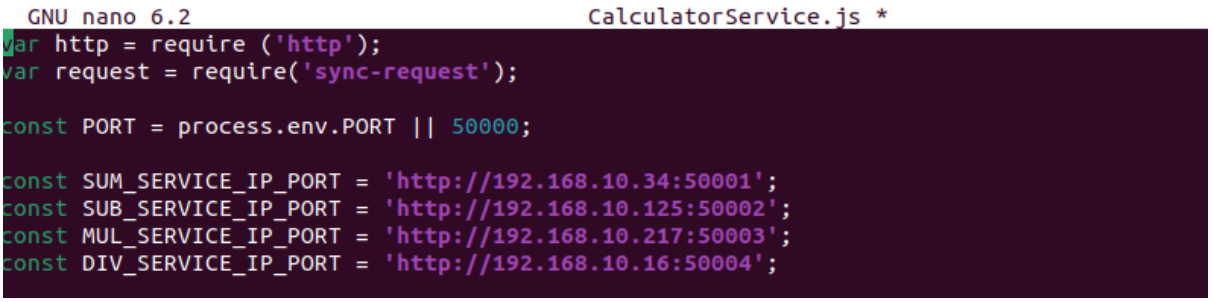
```
$ sudo apt install npm
```

```
$ sudo apt install curl
```

```
$ npm install syn-request
```

Now, we will have to modify the CalculatorService.js by changing the listening port from 80 to 50000.

We will have to modify the IP addresses as well :



Port of sumService: 50001

Port of subService: 50002

Port of mulService: 50003

Port of divService: 50004

HTTP is bidirectional so let's not forget to add new rules to the security groups : Entrée et Sortie as follows :

☐	Direction	Ether Type	IP Protocol	Port Range	Remote IP Prefix	Remote Security Group	Description	Actions
☐	Sortie	IPv4	Tous	Tous	0.0.0.0/0	-	-	Supprimer une Règle
☐	Sortie	IPv6	Tous	Tous	::/0	-	-	Supprimer une Règle
☐	Entrée	IPv4	Tous	Tous	-	default	-	Supprimer une Règle
☐	Entrée	IPv4	ICMP	Tous	0.0.0.0/0	-	-	Supprimer une Règle
☐	Entrée	IPv4	TCP	22 (SSH)	0.0.0.0/0	-	-	Supprimer une Règle
☐	Entrée	IPv4	TCP	50000 - 50004	0.0.0.0/0	-	-	Supprimer une Règle
☐	Entrée	IPv6	Tous	Tous	-	default	-	Supprimer une Règle

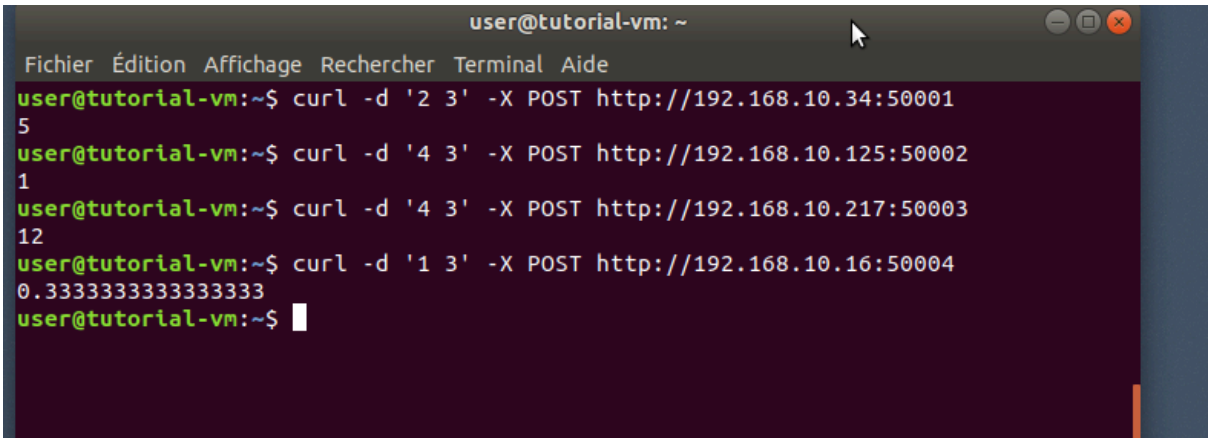
Affichage de 7 éléments

Finally, we have to give the frontend machine a floating IP to be able to communicate with it from outside the private network. We can then ping it from our separate ubuntu VM.

We can now request the services with the command curl :

First, All VMs must be in listening mode via the command node <Service>.js

Locally, from the VM **:**



⇒ we can see that the requests are working

Now, from the outside using the floating IP :

```
osboxes@osboxes:~$ curl -d "(5+6)*2" -X POST http://192.168.37.152:50000
result = 22

osboxes@osboxes:~$
```

Finally, the client received the answer.

Using a text editor (e.g. nano), and without stopping the application, change the source code of one µService (e.g. adding some display instructions in between coding lines) :

When modifying the source code without stopping the application (e.g., adding a console.log) we notice that it receives the request, performs the operation, but fails to transmit the result back.

4. Expected work for objectives 10 and 11

Part two: Deployment of the target topology

In this section, we will deploy a network topology described as follows :

- Sub-Network 1 (IP = 192.168.1.0/24)
  - 1 virtual machine hosting the calculator front end service
  - 2 network appliances
    - § 1 router (IP 192.168.1.254) that provides access to end users
    - § 1 router (IP 192.168.1.253) that makes the bridge between the public network and sub-network 1
- Sub-Network 2 (IP = 192.168.2.0/24)
  - 4 virtual machines that host the services implementing the four arithmetic operations
  - 1 network appliance
    - § 1 router (192.168.2.254) that makes the bridge between the 2 sub-networks

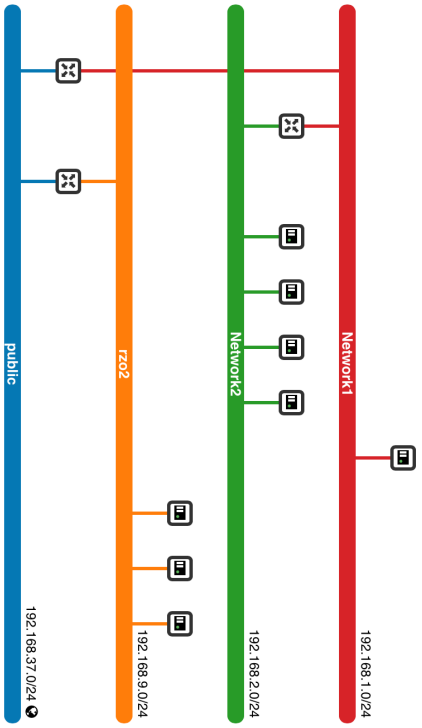
In order to do that, we will use a python script and json descriptor file.

The source codes can be found in the github :

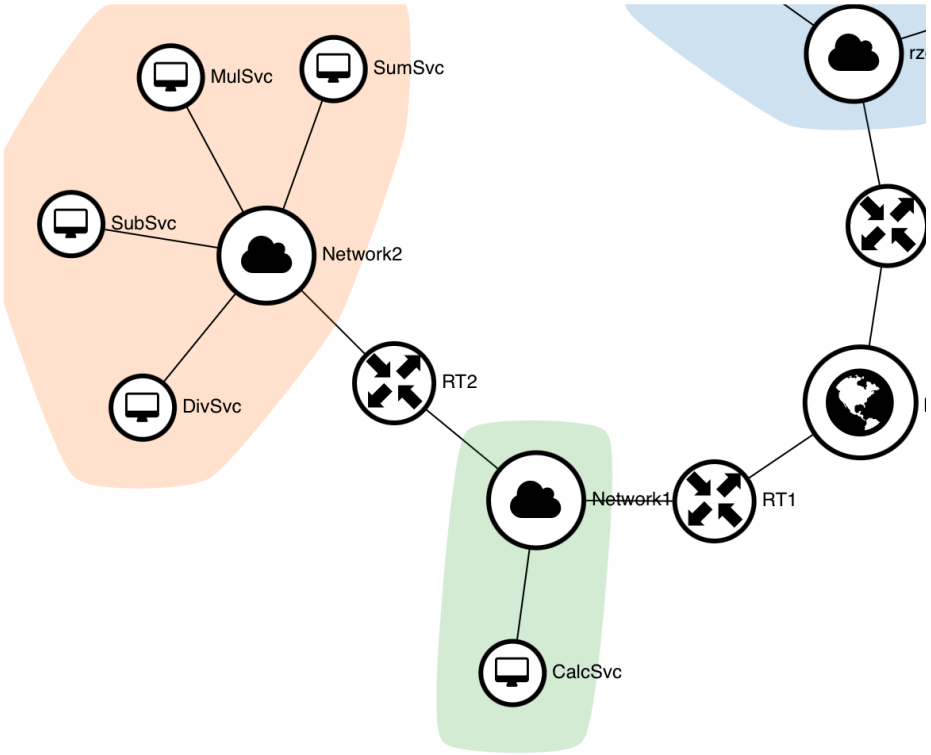
https://github.com/LamiaaHousni/Cloud.git

We managed to create networks, sub-networks, instances, routers and interfaces via the python script. The only thing that did not work was adding the interface of sub-net 1 to the RT2 (error : the IP address is already in use) so we had to add another interface (different IP address) manually via OpenStack.

Here is the resulted topology : (Only Network 1 - Network 2 - Public)



Graphic View :



Now, we will configure the VM as we did in the previous part.

- Test the connectivity between the VM hosting the calculator front end and any machine from the ones hosting the arithmetic operations services. What do you observe? Elaborate on this?

We noticed that the end machine and the calculating machine can't communicate, in fact they don't belong to the same network.

In order to resolve the problem we will add two routes :

- 1 route that redirect the front end machine to the network 2 any time it tries to communicate with one of the 4 machines :

```
sudo route add -net 192.168.2.0/24 gw 192.168.1.254
```

- 1 default route in network 1 :

```
sudo route add -net default gw 192.168.1.253
```

Finally, after modifying the IP addresses in the source file the calculator service works properly.

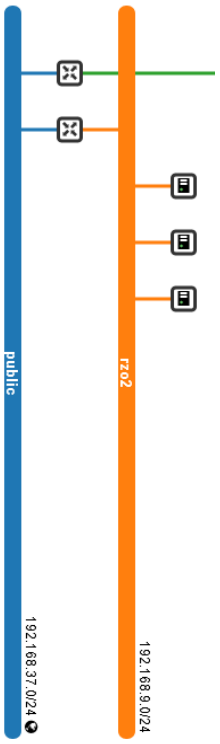
Lab 2 : Orchestrating services in hybrid cloud/edge environment

Part 1: Cloud infrastructure setup

This part aims to set up the Kubernetes cluster along with its 3 running nodes.

First, we will create a new network topology as follows :

- 1 Master Node (Ubuntu_20 cloud image & Medium Flavor)
- 2 Worker nodes (Ubuntu_20 cloud image & Small2 Flavor)

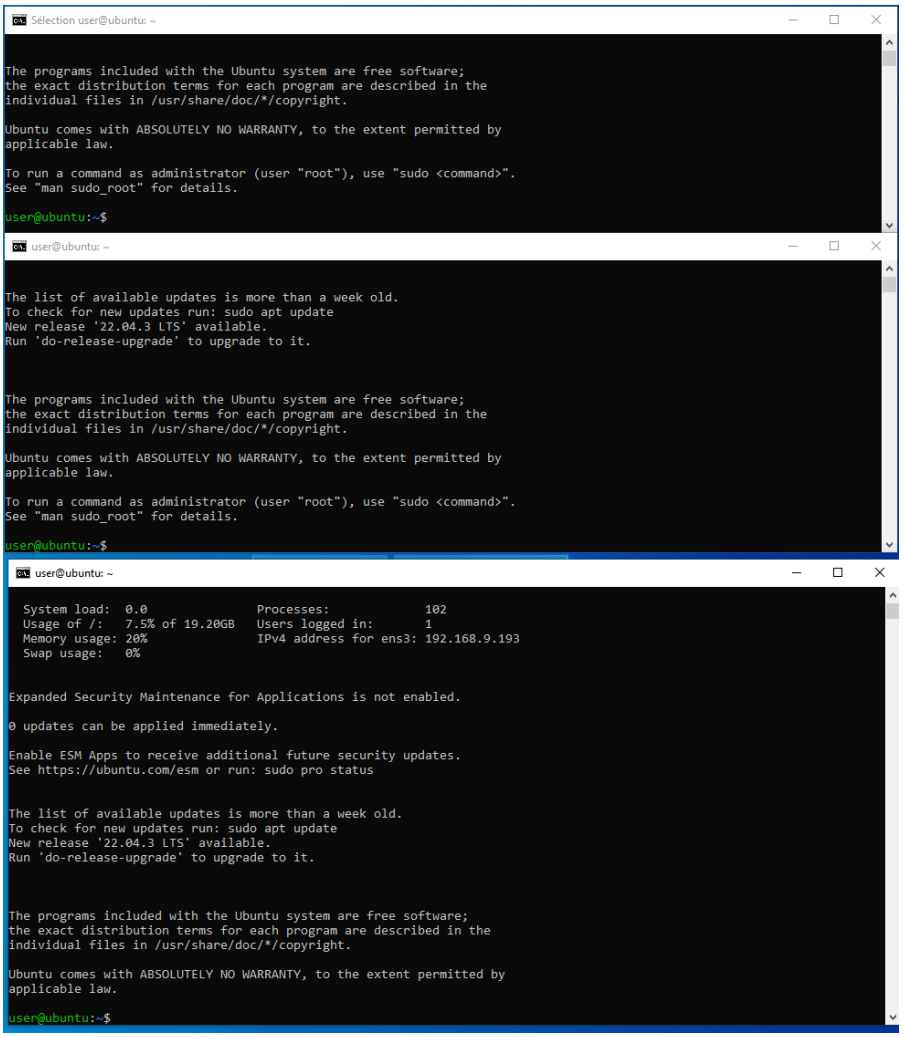


Now, we will generate 3 floating IP addresses and associate them to the VMS in order to setup ssh connexions later.

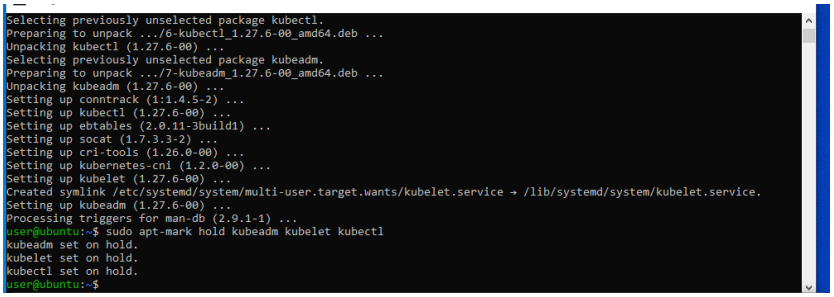
☐	192.168.37.4	Worker_1 192.168.9.209	public	Active	Dissocier
☐	192.168.37.141	Worker_2 192.168.9.144	public	Active	Dissocier
☐	192.168.37.109	Master 192.168.9.128	public	Active	Dissocier

On each machine we will create a new “user” (with password “user”) and confirm default configuration following the tutorial. (kubernetes and docker configuration)

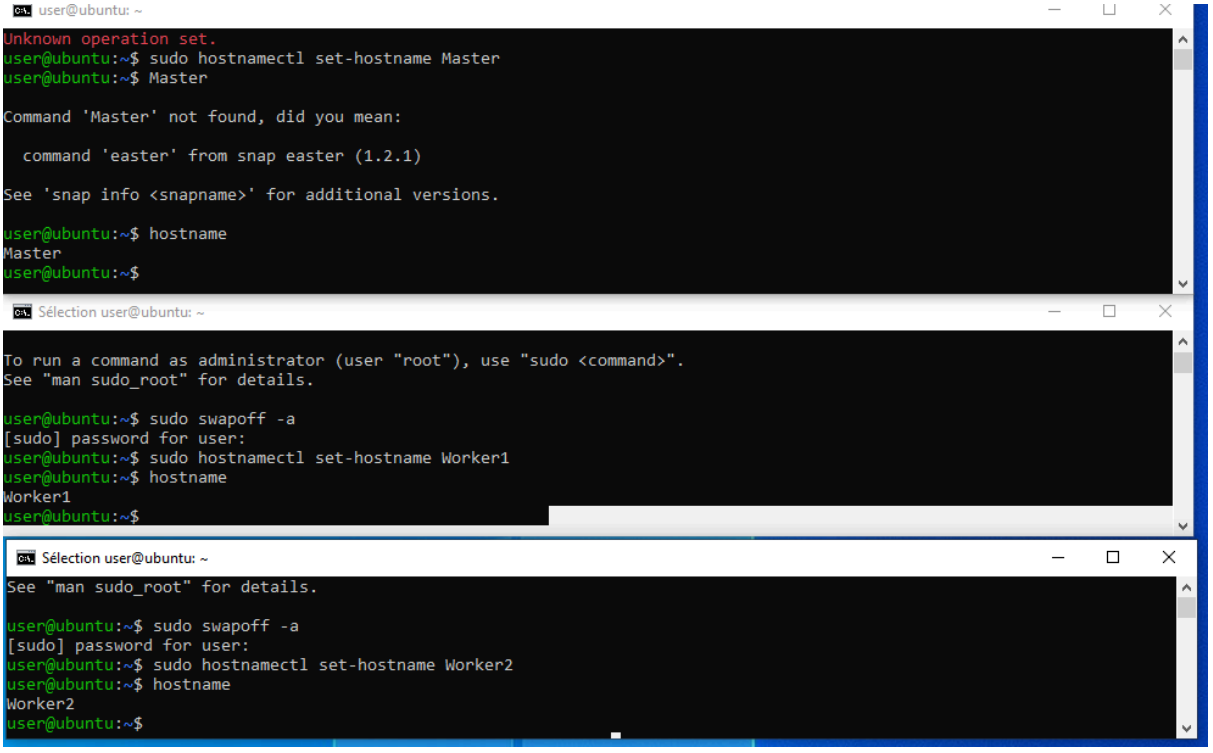
SSH Connection for the 3 nodes :



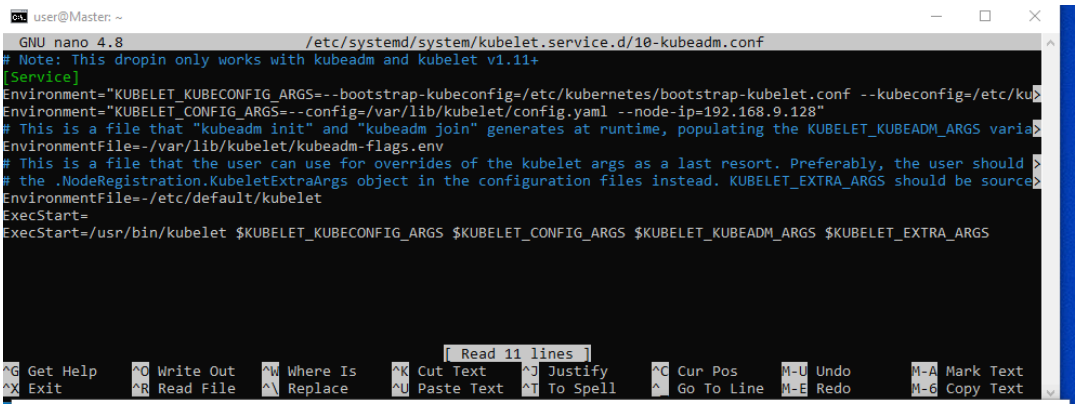
Install kubernetes components and mark them hold to keep them running anytime your system is up :



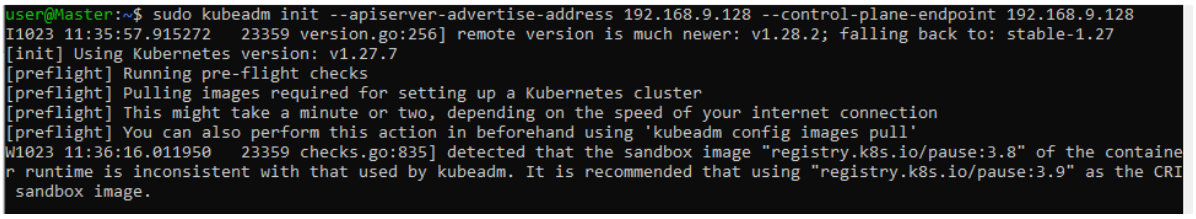
Change of hostnames to avoid confusion:



Before starting the cluster, we need to edit the kubelet config file as follows by adding the flag - - node-ip to the KUBELET_CONFIG_ARGS with the IP of our ens3 interface :

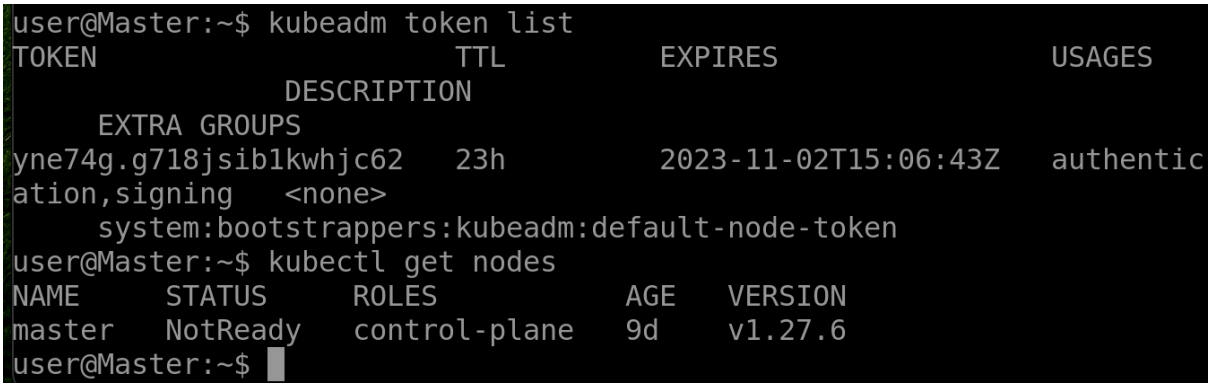


Now we will initialize our master :



Now, we want to join the cluster. We will need the token and token certificate

Command to obtain the token :



Command to obtain the certificat token :

```
user@Master:~$ openssl x509 -pubkey -in /etc/kubernetes/pki/ca.c
rt | openssl rsa -pubin -outform der 2>/dev/null | openssl dgst
-sha256 - -hex  | sed 's/^.* //'
92708ae06e8c6258ef3cc00ebf987d3d2a20c6691ac2964307391cf08469bd6b
user@Master:~$
```

Now the Worker1 will join the cluster :

```
Last login: Wed Nov  1 16:03:54 2023 from 10.10.40.245
user@Worker1:~$ sudo kubeadm join 192.168.9.128:6443 --token yne74g.g718jsib1
kwhjc62 --discovery-token-ca-cert-hash sha256:92708ae06e8c6258ef3cc00ebf987d3
d2a20c6691ac2964307391cf08469bd6b
[sudo] password for user:
[preflight] Running pre-flight checks
[preflight] Reading configuration from the cluster...
[preflight] FYI: You can look at this config file with 'kubectl -n kube-syste
m get cm kubeadm-config -o yaml'
[kubelet-start] Writing kubelet configuration to file "/var/lib/kubelet/confi
g.yaml"
[kubelet-start] Writing kubelet environment file with flags to file "/var/lib
/kubelet/kubeadm-flags.env"
[kubelet-start] Starting the kubelet
[kubelet-start] Waiting for the kubelet to perform the TLS Bootstrap...

This node has joined the cluster:
* Certificate signing request was sent to apiserver and a response was receiv
ed.
* The Kubelet was informed of the new secure connection details.

Run 'kubectl get nodes' on the control-plane to see this node join the cluste
r.

user@Worker1:~$
```

Checking on the master node if the worker1 has joined the cluster :

```
user@Master:~$ kubectl get nodes
NAME      STATUS     ROLES           AGE     VERSION
master    NotReady   control-plane   9d      v1.27.6
worker1   NotReady   <none>          3m17s   v1.27.6
user@Master:~$
```

⇒ Worker1 has joined the cluster

Same for Worker2:

```
user@Master:~$ kubectl get nodes
NAME      STATUS     ROLES           AGE     VERSION
master    NotReady   control-plane   9d      v1.27.6
worker1   NotReady   <none>          7m36s   v1.27.6
worker2   NotReady   <none>          24s     v1.27.6
user@Master:~$
```

What do you notice?

We notice that STATUS of the nodes in : Not Ready

The networking needs one more step to properly work. Kubernetes is a container orchestration tool which doesn't natively provide networking. However, it exposes an interface called *Container Networking Interface* which can be connected to components that can provide networking management. In our case we will be using *Calico*. To apply it on the cluster, we run the following command on the master node:

```
$ kubectl apply -f https://docs.projectcalico.org/manifests/calico.yaml
```

Before running the command :

```
*** System restart required ***
Last login: Wed Nov  1 16:17:56 2023 from 10.10.40.244
user@Master:~$ kubectl get nodes
NAME      STATUS     ROLES           AGE     VERSION
master    NotReady   control-plane   9d      v1.27.6
worker1   NotReady   <none>          7m36s   v1.27.6
worker2   NotReady   <none>          24s     v1.27.6
user@Master:~$ kubectl get pods -n calico
No resources found in calico namespace.
user@Master:~$ kubectl get pods -A
NAMESPACE     NAME                                READY   STATUS    RESTAR
TS   AGE
kube-system   coredns-5d78c9869d-6tr7x            0/1     Pending   0
     9d
kube-system   coredns-5d78c9869d-vt8px            0/1     Pending   0
     9d
kube-system   etcd-master                         1/1     Running   0
     9d
kube-system   kube-apiserver-master               1/1     Running   0
     9d
kube-system   kube-controller-manager-master      1/1     Running   0
     9d
kube-system   kube-proxy-skzqg                    1/1     Running   0
     2m59s
kube-system   kube-proxy-tdswq                    1/1     Running   0
     10m
kube-system   kube-proxy-wccfj                    1/1     Running   0
     9d
kube-system   kube-scheduler-master               1/1     Running   0
     9d
```

⇒ we notice that the nodes are not ready and their status is pending

After applying the command :

```
user@Master:~$ kubectl get pods -A
NAMESPACE     NAME                                       READY   STATUS    RESTARTS   AGE
kube-system   calico-kube-controllers-85578c44bf-d4rk5   1/1     Running   0          3m9s
kube-system   calico-node-6wq56                          1/1     Running   0          3m9s
kube-system   calico-node-gpg5m                          1/1     Running   0          3m9s
kube-system   calico-node-mg4kw                          1/1     Running   0          3m9s
kube-system   coredns-5d78c9869d-6tr7x                   1/1     Running   0          9d
kube-system   coredns-5d78c9869d-vt8px                   1/1     Running   0          9d
kube-system   etcd-master                                1/1     Running   0          9d
kube-system   kube-apiserver-master                      1/1     Running   0          9d
kube-system   kube-controller-manager-master             1/1     Running   0          9d
kube-system   kube-proxy-skzqg                           1/1     Running   0          7m19s
kube-system   kube-proxy-tdswq                           1/1     Running   0          14m
kube-system   kube-proxy-wccfj                           1/1     Running   0          9d
kube-system   kube-scheduler-master                      1/1     Running   0          9d
user@Master:~$ kubectl get nodes -o wide
NAME      STATUS   ROLES           AGE     VERSION   INTERNAL-IP     EXTERNAL-IP   OS-IMAGE             KERNEL-VERSION      CONTAINER-RUNTIME
master    Ready    control-plane   9d      v1.27.6   192.168.9.128   <none>        Ubuntu 20.04.6 LTS   5.4.0-163-generic   containerd://1.7.2
worker1   Ready    <none>          14m     v1.27.6   192.168.9.209   <none>        Ubuntu 20.04.6 LTS   5.4.0-163-generic   containerd://1.7.2
worker2   Ready    <none>          7m25s   v1.27.6   192.168.9.144   <none>        Ubuntu 20.04.6 LTS   5.4.0-163-generic   containerd://1.7.2
user@Master:~$
```

⇒ The nodes are ready and running

Kuberneetes cluster all set up 👍

Kubernetes overview Hands-on:

We are going to discover ClusterIP and NodePort services and understand why they are important. Let us first deploy a simple app which returns hello world to the client using fastapi.

Docker login and secret creation :

```
user@Master:~$ docker login
Login with your Docker ID to push and pull images from Docker Hub. If y
ave a Docker ID, head over to https://hub.docker.com to create one.
Username: yxos
Password:
WARNING! Your password will be stored unencrypted in /home/user/.docker
on.
Configure a credential helper to remove this warning. See
https://docs.docker.com/engine/reference/commandline/login/#credentials

Login Succeeded
user@Master:~$ kubectl create secret docker-registry repo-secret --do
cker-username=yxos --docker-password=Dockwhizzer007! --docker-email=j
ean.cedricsanou@gmail.com
secret/repo-secret created
user@Master:~$ kubectl get secret repo-secret --output=yaml
apiVersion: v1
data:
  .dockerconfigjson: eyJhdXRocyI6eyJodHRwczovL2luZGV4LmRvY2tlci5pby92
MS8iOnsidXNlcm5hbWUiOiJ5eG9zIiwicGFzc3dvcmQiOiJEb2Nrd2hpenplcjAwNyEiL
CJlbWFpbCI6ImplYW4uY2Vkcmljc2Fub3VAZ21haWwuY29tIiwiYXV0aCI6ImVYaHZjen
BFYjJOcmQyaHBlbnBsY2pBd055RT0ifX19
```

● **ClusterIP**

First, we will download the ClusterIP_service folder using the command scp.

Then, run :

```
user@Master:~$ kubectl apply -f ./ClusterIP
deployment.apps/fastapi-app created
Warning: annotation "kubernetes.io/ingress.class" is deprecated, please us
e 'spec.ingressClassName' instead
ingress.networking.k8s.io/fastapi-app-ingress created
service/fastapi-app-clusterip-service created
user@Master:~$ kubectl get pods -o wide
NAME                          READY   STATUS              RESTARTS   AGE
  IP       NODE      NOMINATED NODE   READINESS GATES
fastapi-app-5476944694-mltt6   0/1     ContainerCreating   0          17s
  <none>   worker1   <none>           <none>
fastapi-app-5476944694-r6h4z   0/1     ContainerCreating   0          17s
  <none>   worker1   <none>           <none>
fastapi-app-5476944694-x9ctt   0/1     ContainerCreating   0          17s
  <none>   worker1   <none>           <none>
user@Master:~$ kubectl get pods -o wide
NAME                          READY   STATUS              RESTARTS   AGE   IP       NODE      NOMINATED NODE   READINESS GATES
fastapi-app-5476944694-mltt6   0/1     ContainerCreating   0          41s   <none>   worker1   <none>           <none>
fastapi-app-5476944694-r6h4z   0/1     ContainerCreating   0          41s   <none>   worker1   <none>           <none>
fastapi-app-5476944694-x9ctt   0/1     ContainerCreating   0          41s   <none>   worker1   <none>           <none>
user@Master:~$
```

We notice that the pods are not ready. To acces them we have to run the command :

```
kubectl get services -o wide
```

After running the command we have :

```
user@Master:~$ kubectl get pods -o wide
NAME                          READY   STATUS    RESTARTS   AGE     IP               NODE      NOMINATE
D NODE   READINESS GATES
fastapi-app-5476944694-mltt6   1/1     Running   0          6m25s   172.16.235.129   worker1   <none>
         <none>
fastapi-app-5476944694-r6h4z   1/1     Running   0          6m25s   172.16.235.130   worker1   <none>
         <none>
fastapi-app-5476944694-x9ctt   1/1     Running   0          6m25s   172.16.235.131   worker1   <none>
         <none>
```

```
user@Master:~$ kubectl get service -o wide
NAME                          TYPE        CLUSTER-IP       EXTERNAL-IP   PORT(S)   AGE     SELECTOR
fastapi-app-clusterip-service   ClusterIP   10.100.102.107   <none>        80/TCP    2m45s   app=fastapi-app
kubernetes                    ClusterIP   10.96.0.1        <none>        443/TCP   9d      <none>
```

⇒ Pods running

Looking for the endpoints where the application is running **:**

```
user@Master:~$ kubectl describe services fastapi-app-clusterip-service
Name:              fastapi-app-clusterip-service
Namespace:         default
Labels:            <none>
Annotations:       <none>
Selector:          app=fastapi-app
Type:              ClusterIP
IP Family Policy:  SingleStack
IP Families:       IPv4
IP:                10.100.102.107
IPs:               10.100.102.107
Port:              <unset>  80/TCP
TargetPort:        5000/TCP
Endpoints:         172.16.235.129:5000,172.16.235.130:5000,172.16.235.131:5000
Session Affinity:  None
Events:            <none>
```

We will try to Curl one of the endpoints **:**

```
user@Master:~$ curl http://172.16.235.129:5000
{"hello":"world"}user@Master:~$
```

⇒ The application is working

Now, from the master node we will run a pod called testpod :

```
user@Master:~$ kubectl run testpod --image=nginx
pod/testpod created
user@Master:~$ kubectl get pods
NAME                          READY   STATUS              RESTARTS   A
GE
fastapi-app-5476944694-mltt6   1/1     Running             0          1
9m
fastapi-app-5476944694-r6h4z   1/1     Running             0          1
9m
fastapi-app-5476944694-x9ctt   1/1     Running             0          1
9m
testpod                        0/1     ContainerCreating   0          9
```

Access inside the pod created **:**

```
user@Master:~$ kubectl exec -it testpod -- /bin/bash
root@testpod:/#
```

Curl from inside the testpod **:**

```
root@testpod:/# curl http://172.16.235.131:5000
{"hello":"world"}root@testpod:/#
```

⇒ We notice that we can run the application even from inside the pod

● Nodeport service

Similarly to before we will download the folder then form the master node run :

```
user@Master:~$ kubectl apply -f ./NodePort
deployment.apps/fastapi-app created
Warning: annotation "kubernetes.io/ingress.class" is deprecated, pleas
e use 'spec.ingressClassName' instead
ingress.networking.k8s.io/fastapi-app-ingress created
service/fastapi-app-service created
user@Master:~$ kubectl get pods -o wide
NAME                           READY   STATUS    RESTARTS   AGE   IP
            NODE      NOMINATED NODE   READINESS GATES
fastapi-app-5476944694-45g2l   1/1     Running   0          18s   172.
16.235.132   worker1   <none>           <none>
fastapi-app-5476944694-49sd9   1/1     Running   0          18s   172.
16.235.133   worker1   <none>           <none>
fastapi-app-5476944694-qx28z   1/1     Running   0          19s   172.
16.235.134   worker1   <none>           <none>
testpod                        1/1     Running   0          49m   172.
16.189.65    worker2   <none>           <none>
```

⇒ Everything running properly.

Now we run :

```
user@Master:~$ kubectl get services -o wide
NAME                 TYPE        CLUSTER-IP     EXTERNAL-IP   PORT(S)        AGE    SELECTOR
fastapi-app-service  NodePort    10.96.178.49   <none>        80:31852/TCP   2m2s   app=fastapi-app
kubernetes           ClusterIP   10.96.0.1      <none>        443/TCP        9d     <none>
user@Master:~$ kubectl describe services fastapi-app-service
Name:                     fastapi-app-service
Namespace:                default
Labels:                   <none>
Annotations:              <none>
Selector:                 app=fastapi-app
Type:                     NodePort
IP Family Policy:         SingleStack
IP Families:              IPv4
IP:                       10.96.178.49
IPs:                      10.96.178.49
Port:                     <unset>  80/TCP
TargetPort:               5000/TCP
NodePort:                 <unset>  31852/TCP
Endpoints:                172.16.235.132:5000,172.16.235.133:5000,172.16.235.134:5000
Session Affinity:         None
External Traffic Policy:  Cluster
Events:                   <none>
```

We notice the value 31852 /TCP attributed to the Nodeport.

This value does not figure on the service_yaml file.

Kubernetes might automatically assign an available port. In this case, the observed value during the deployment is different from what's specified in the YAML file.

Curl :

```
user@Master:~$ curl http://172.16.235.132:5000
{"hello":"world"}user@Master:~$
```

(We should’ve teste with the NodePort 31852)

Part 2: Services deployment in the hybrid cloud/edge

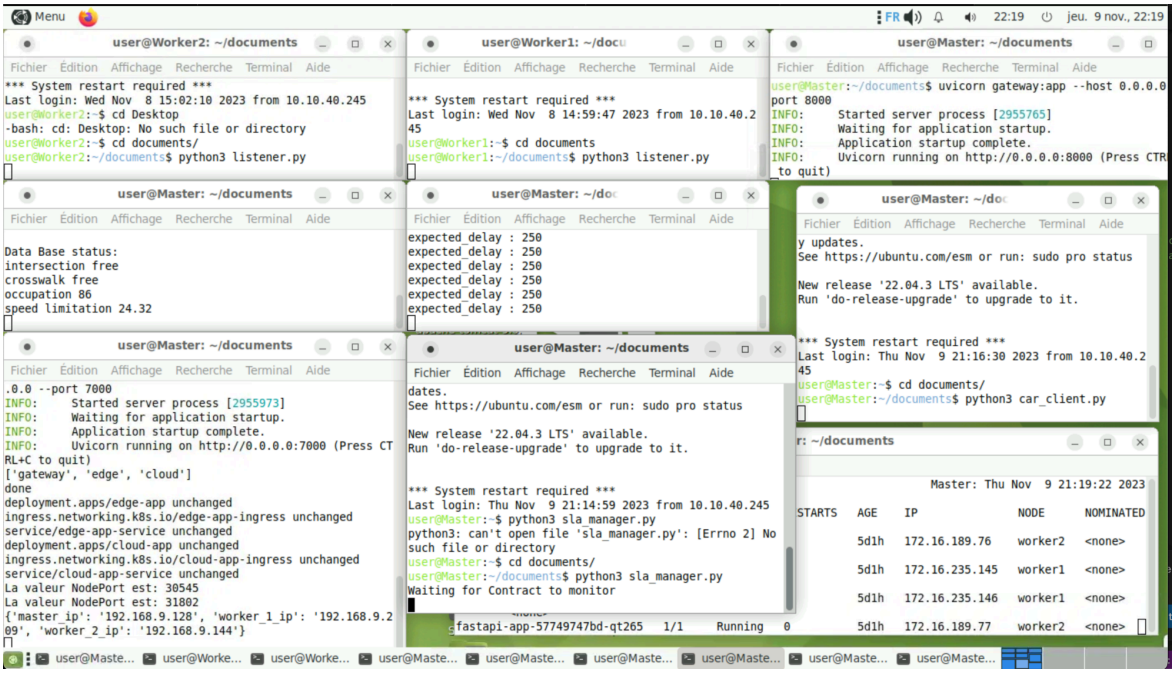
Ressources:

In this section, the first thing to do is to download the resources folder for the different node. We can do this using the command scp.

Then, we will continue by modifying the IP addresses in the data.json files on each node :

```
GNU nano 4.8                                    data.json
{
    "master_ip":"192.168.9.128",
    "worker_1_ip":"192.168.9.209",
    "worker_2_ip":"192.168.9.144"
}
```

By following the tutorial, we obtain the terminals as follows :



Our setup did not work, in fact the controller sends only the 2 first nodePorts and then crushes.

We tried investigating the problem and redoing the setup from scratch, but the problem persists.

We think it might be due to some network issue because we were using a VPN connection