

Cloud Computing



Chanfreau Cedric
Boukouiss Samia
5 ISS

15/10/2024

Lab 1: Introduction to Cloud Hypervisors

Theoretical part

Objectives 1 to 3:

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

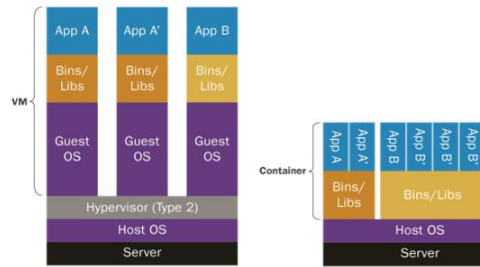


Figure 1: VM vs CT

We will compare the two types of hosts based on two perspectives: from an application developer's point of view, and from an infrastructure administrator point of view.

From an application developer's :

	Virtual Machine	Container
Virtualization cost, taking into consideration memory size and CPU	It requires more memory, as each VM includes its own OS. It's more expensive to virtualize because it emulates an entire machine on the hardware level.	It requires less memory because the CT doesn't duplicate the OS and the application is bundled to run across various environments. This makes it 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. So, there is 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)	More secure because there is a strong isolation between	Security flaws because containers share the same kernel and libraries as the

	applications (each VM runs its own complete OS).	host system (they are more vulnerable to attacks and exploits)
Performances (response time)	Higher response time because for each operation we have to regenerate the environment.	Lower response time, startup in ms.
Tooling for the continuous integration support	No widespread development tools.	It comes with development kits.
Flexibility, dynamicity	Difficult to modify disk size and allocated resources for a VM, and transferring programs requires more effort.	Easy to adjust allocated resources for a container or application, with minimal code needed to transfer and deploy work.

From an infrastructure administrator :

1- Developer point of view :

Virtual Machines	Containers
VM, though bulkier and less flexible, are useful for tasks requiring full machine emulation or precise network control. However, due to their high costs, slow boot times, and excessive resource demands, they are often an impractical choice for most development needs.	CT offer developers portability and flexibility, enabling cost-effective and efficient application hosting. Their lightweight nature allows for resource sharing, quick startup, and response times, making them ideal for testing and deployment. Additionally, bundled development kits provide extra benefits.

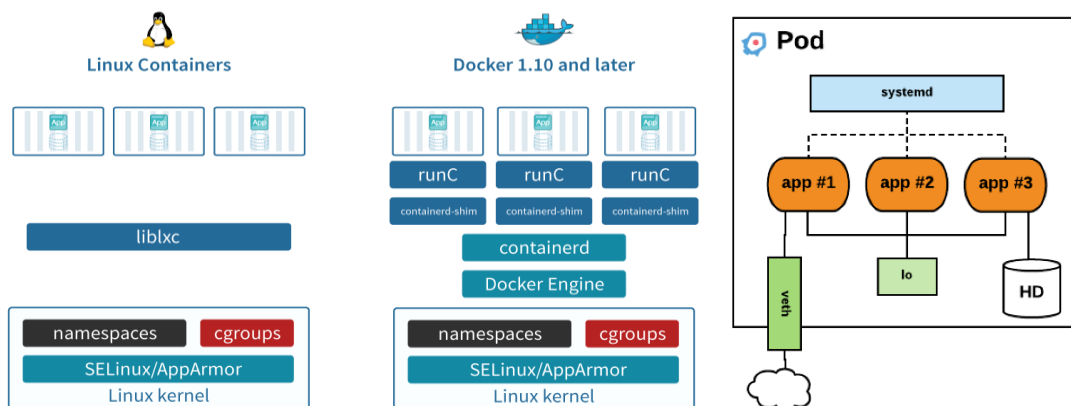
2- System Administrator point of view :

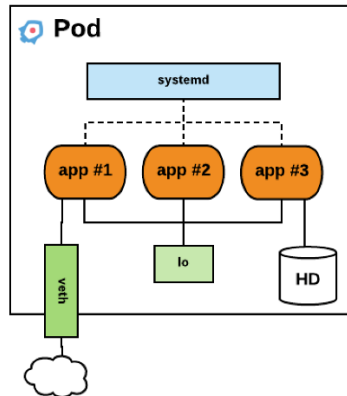
Virtual Machines	Containers
VM are valued by system administrators for their complete isolation, which enhances security and independence, and their unique OS that provide specialized tools. Although they are resource-intensive, this is typically manageable for administrators overseeing large systems with sufficient resources.	CT present notable challenges for system administrators due to security risks, limited control over hardware, and restricted network configurations. These issues make CT less suitable for system administration tasks.

2. Similarities and differences between the existing CT types

Different CT technologies are available in the market (e.g. LXC/LXD, Docker, Rocket, OpenVZ, runC, containerd, systemd-nspawn). Their respective positioning is not obvious, but comparative analyses are available online such as:

- LXC is an operating system-level virtualization tool. It allows running multiple isolated Linux operating system instances on a single Linux host, while sharing the same kernel.
- Docker is a containerization platform that allows applications to run in isolation. Unlike LXC, Docker focuses specifically on containerizing applications, rather than entire operating systems.





We define here the criteria we will use to compare the container technologies (CT) :

- **Application Isolation and Resources (Multi-tenancy):**

It refers to the architecture of the server hosting containers, where a single instance of the host OS supports multiple tenants (containers).

- **Containerization Level (e.g., Operating System, Application):**

Containerization packages software and its dependencies, enabling it to run anywhere.

- **Tooling (e.g., API, Continuous Integration, Service Composition):**

It refers to the tools offered with the container service, such as development kits, migration tools, and custom settings.

Technology	Application Isolation	Containerization level	Tooling
LXC	Light Isolation: Although containers share the core, they remain isolated from each other, ensuring adequate security and stability.	OS level containers, allowing kernel sharing with the host.	LxC offers a CLI for container management and supports APIs.
Docker	Enhanced Container Isolation provides an	Operating system level containers, allowing	Docker is well-known for its user-friendly CLI

	additional layer of security to prevent malicious workloads running in containers from compromising Docker Desktop or the host.	kernel sharing with the host.	and API. It has strong CI/CD support with Docker Compose and Kubernetes integration. Docker Compose is useful for service composition.
Rocket	RKT provides application isolation using proven mechanisms such as Control Groups (Cgroups) and SELinux, ensuring that containers run in protected environments and limit their access to system resources.	Application-level containers, with a layered architecture allowing for flexibility.	It features a CLI for pod-based deployments and integrates seamlessly with systemd for service.
OpenVZ	Good isolation even if containers share the kernel.	Application level	It offers a variety of management tools.
runC	runC is a CLI tool for spawning and running containers on Linux according to the OCI specification.	Application-level containers, integrated in larger systems.	It operates with a command-line interface and adheres to OCI specifications for managing container lifecycles.

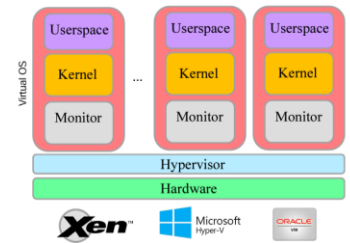
<https://rocket.readthedocs.io/en/latest/Documentation/rkt-vs-other-projects/>

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

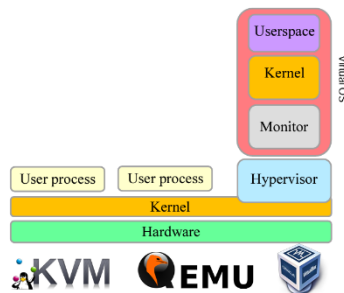
There are two main types of hypervisors:

- Type 1 (or “bare metal”)
- Type 2 (or “hosted”)

A Type 1 hypervisor runs directly on the host's hardware, functioning like a lightweight operating system and it is preferred in enterprise and production environments for its performance and security.



While a Type 2 hypervisor operates as a software layer on an existing operating system, like other applications and it is commonly used in desktop and development settings for its ease of use.



	Architecture	Performance	Use cases
Type 1 (OpenStack)	It operates directly on the physical host machine's bare-metal hardware without the need for an underlying OS.	It usually provides better performance by directly accessing hardware resources, eliminating the overhead of an operating system layer.	It is commonly used in enterprise data centers and cloud environments where performance, scalability, and resource isolation are critical.

Type 2 (VirtualBox)	It runs on top of an existing OS and operates as applications or processes within that traditional OS.	It generally has more overhead than Type 1 hypervisors because it depends on the host OS to manage hardware resources.	It is commonly used in desktop or developer environments where performance is less critical.

Practical part

Objectives 4 to 7 :

In this part, we will use the VirtualBox hypervisor (type 2) in NAT mode and set up the network to enable two-way of communication with the outside.

First part: Creating and configuring a VM

- Open VirtualBox
- Unzip archive
- Create and configure a new VM
- Launch the VM

Second part: Testing the VM connectivity

- a) VM Connection

Once logged in, we used the *ifconfig* command in the VM terminal to identify the assigned IP address:


```
osboxes@osboxes: ~/Desktop
osboxes@osboxes:~/Desktop$ ifconfig
docker0: flags=4099<UP,BROADCAST,MULTICAST> mtu 1500
    inet 172.17.0.1 netmask 255.255.0.0 broadcast 172.17.255.255
    ether 02:42:10:08:05:9b txqueuelen 0 (Ethernet)
    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

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::e684:9549:3e7:5f73 prefixlen 64 scopeid 0x20<link>
    ether 08:00:27:7f:16:43 txqueuelen 1000 (Ethernet)
    RX packets 28284 bytes 42401574 (42.4 MB)
    RX errors 0 dropped 0 overruns 0 frame 0
    TX packets 1435 bytes 120516 (120.5 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 (Local Loopback)
    RX packets 241 bytes 19981 (19.9 KB)
    RX errors 0 dropped 0 overruns 0 frame 0
```

NAT Address: 172.17.0.1

Virtual Machine Address: 10.0.2.15

Loopback address : 127.0.0.1

```
Windows PowerShell
PS U:\>
PS U:\> Get-NetIPAddress -AddressFamily IPV4

IPAddress      : 192.168.56.1
InterfaceIndex : 9
InterfaceAlias : VirtualBox Host-Only Network
AddressFamily  : IPv4
Type           : Unicast
PrefixLength   : 24
PrefixOrigin   : Manual
SuffixOrigin   : Manual
AddressState   : Preferred
ValidLifetime  :
PreferredLifetime :
SkipAsSource   : False
PolicyStore    : ActiveStore

IPAddress      : 10.1.5.89
InterfaceIndex : 8
InterfaceAlias : Ethernet
```

Host Machine associated to VirtualBox Host Only-Network: 192.168.56.1

Address Ethernet Interface of Host: 10.1.5.89

b) Connectivity Verification

In this part, we tested the different connectivity between each component:

- Vm ↔ Host Machine: Ping Ok

```
osboxes@osboxes:~/Desktop$ ping 192.168.56.1
PING 192.168.56.1 (192.168.56.1) 56(84) bytes of data:
64 bytes from 192.168.56.1: icmp_seq=1 ttl=127 time=1.44 ms
64 bytes from 192.168.56.1: icmp_seq=2 ttl=127 time=1.23 ms
64 bytes from 192.168.56.1: icmp_seq=3 ttl=127 time=1.40 ms
64 bytes from 192.168.56.1: icmp_seq=4 ttl=127 time=1.12 ms
64 bytes from 192.168.56.1: icmp_seq=5 ttl=127 time=0.885 ms
^C
```

- Host Machine ⇔ Nat / VM: Ping Failed

```
PS 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.

Statistiques Ping pour 10.0.2.15:
    Paquets : envoyés = 2, reçus = 0, perdus = 2 (perte 100%),
Ctrl+C
PS U:\> ping 172.17.0.1

Envoi d'une requête 'Ping' 172.17.0.1 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.
```

To conclude those tests, it was not possible to ping the host machine with the VM because of the NAT that is between, and the address is not routable. Whereas the opposite was possible.

To resolve this issue, a solution would be to use a port forwarding rule. Allowing to ensure the communication by redirecting request from host to any port configured inside the VM.

Third part: Set up the “missing” connectivity

To enable communication between the host machine and the VM in NAT mode, we can configure port forwarding in VirtualBox

By specifying that all data arriving on the physical machine at a specific port (1234) is destined for the VM port 22.

Règles de redirection de ports					
Nom	Protocole	IP hôte	Port hôte	IP invité	Port invité
Rule 1	TCP		1234		22

After this configuration, we installed openssh on VM to test thanks PuTTY that the connection was well established.

```
PS U:\> ssh osboxes@localhost -p 1234
osboxes@localhost's password:
Welcome to Ubuntu 22.04 LTS (GNU/Linux 5.15.0-25-generic x86_64)

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

332 updates can be applied immediately.
146 of these updates are standard security updates.
To see these additional updates run: apt list --upgradable

New release '24.04.1 LTS' available.
Run 'do-release-upgrade' to upgrade to it.

Last login: Tue Oct  1 09:03:28 2024 from 127.0.0.1
osboxes@osboxes:~$
```

It is possible now to ping the VM with any machine on the network.

```
PS U:\> ping 127.0.1.1

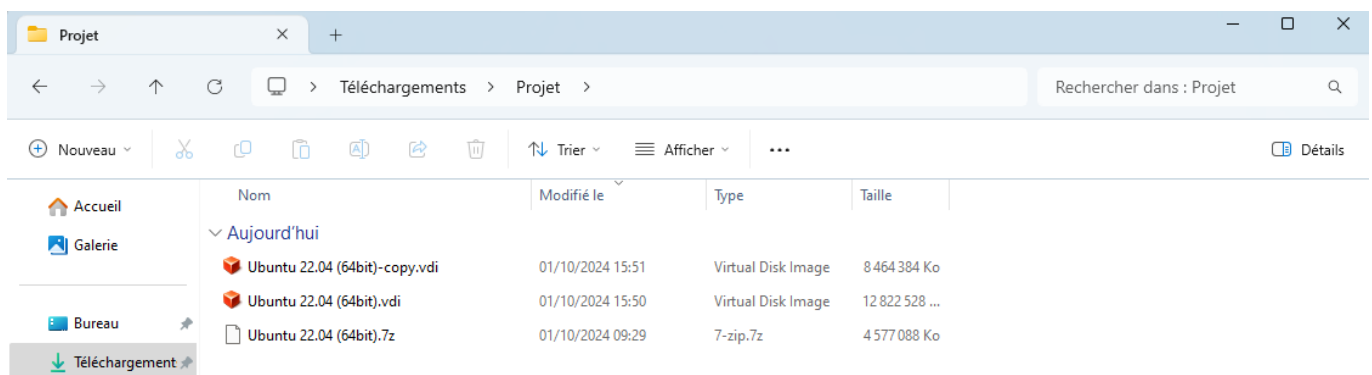
Envoi d'une requête 'Ping' 127.0.1.1 avec 32 octets de données :
Réponse de 127.0.1.1 : octets=32 temps<1ms TTL=128
Réponse de 127.0.1.1 : octets=32 temps<1ms TTL=128
Réponse de 127.0.1.1 : octets=32 temps<1ms TTL=128
Réponse de 127.0.1.1 : octets=32 temps<1ms TTL=128

Statistiques Ping pour 127.0.1.1:
    Paquets : envoyés = 4, reçus = 4, perdus = 0 (perte 0%),
    Durée approximative des boucles en millisecondes :
        Minimum = 0ms, Maximum = 0ms, Moyenne = 0ms
```

Fourth part: VM duplication

To create a new clone with the same disk file, here is the command we used:

```
PS C:\Program Files\Oracle\VirtualBox> .\VBoxManage.exe clonemedium "C:\Users\chanfreau\Downloads\Projet\Ubuntu 22.04 (64bit).vdi" "C:\Users\chanfreau\Downloads\Projet\Ubuntu 22.04 (64bit)-copy.vdi"
0%
```



1. Docker Containers Setup:

- Update existing list of packages:

```
osboxes@osboxes:~/Desktop$ sudo apt update
[sudo] password for osboxes:
Get:1 https://download.docker.com/linux/ubuntu jammy InRelease [48.8 kB]
Get:2 http://security.ubuntu.com/ubuntu jammy-security InRelease [129 kB]
Hit:3 http://us.archive.ubuntu.com/ubuntu jammy InRelease
Get:4 http://us.archive.ubuntu.com/ubuntu jammy-updates InRelease [128 kB]
Get:5 https://download.docker.com/linux/ubuntu jammy/stable amd64 Packages [40.7 kB]
Get:6 http://us.archive.ubuntu.com/ubuntu jammy-backports InRelease [127 kB]
Get:7 http://security.ubuntu.com/ubuntu jammy-security/main amd64 Packages [1,84
```

- Install prerequisite packages which let apt use packages over HTTPS:

```
osboxes@osboxes:~/Desktop$ sudo apt install apt-transport-https ca-certificates
curl software-properties-common
Reading package lists... Done
Building dependency tree... Done
Reading state information... Done
The following additional packages will be installed:
  libcurl4 python3-software-properties software-properties-gtk
  ubuntu-advantage-tools ubuntu-pro-client
```

- Add the GPG key for the official Docker + Docker repository to APT sources:

```
osboxes@osboxes:~/Desktop$ curl -fsSL https://download.docker.com/linux/ubuntu/gpg | sudo gpg --dearmor -o /usr/share/keyrings
docker-archive-keyring.gpg
osboxes@osboxes:~/Desktop$ echo "deb [arch=$(dpkg --print-architecture) signed-by=/usr/share/keyrings/docker-archive-keyring.g
pg] https://download.docker.com/linux/ubuntu $(lsb_release -cs) stable" | sudo tee /etc/apt/sources.list.d/docker.list > /dev/n
```

- Update existing list of packages again for the addition to be recognized:

```
osboxes@osboxes:~/Desktop$ sudo apt update
[sudo] password for osboxes:
Get:1 https://download.docker.com/linux/ubuntu jammy InRelease [48.8 kB]
Get:2 http://security.ubuntu.com/ubuntu jammy-security InRelease [129 kB]
Hit:3 http://us.archive.ubuntu.com/ubuntu jammy InRelease
Get:4 http://us.archive.ubuntu.com/ubuntu jammy-updates InRelease [128 kB]
Get:5 https://download.docker.com/linux/ubuntu jammy/stable amd64 Packages [40.7 kB]
Get:6 http://us.archive.ubuntu.com/ubuntu jammy-backports InRelease [127 kB]
Get:7 http://security.ubuntu.com/ubuntu jammy-security/main amd64 Packages [1,84
```

- Make sure we will install from Docker repository instead of Ubuntu repository:
- Install Docker + Check it is running:

```
● docker.service - Docker Application Container Engine
   Loaded: loaded (/lib/systemd/system/docker.service; enabled; vendor preset: enabled)
   Active: active (running) since Tue 2024-10-01 10:08:12 EDT; 1min 57s ago
 TriggeredBy: ● docker.socket
   Docs: https://docs.docker.com
  Main PID: 12225 (dockerd)
    Tasks: 8
   Memory: 7.7M
      CPU: 715ms
  CGroup: /system.slice/docker.service
          └─12225 /usr/bin/dockerd -H fd:// --containerd=/run/containerd/containerd.sock

Oct 01 10:08:11 osboxes dockerd[12225]: time="2024-10-01T10:08:11.593695091-04:00" level=info msg="detected 127.0.0.53 nameser
Oct 01 10:08:11 osboxes dockerd[12225]: time="2024-10-01T10:08:11.869972651-04:00" level=info msg="[graphdriver] using prior s
Oct 01 10:08:11 osboxes dockerd[12225]: time="2024-10-01T10:08:11.908798973-04:00" level=info msg="Loading containers: start."
Oct 01 10:08:12 osboxes dockerd[12225]: time="2024-10-01T10:08:12.432100563-04:00" level=info msg="Default bridge (docker0) is
Oct 01 10:08:12 osboxes dockerd[12225]: time="2024-10-01T10:08:12.537547171-04:00" level=warning msg="error locating sandbox i
Oct 01 10:08:12 osboxes dockerd[12225]: time="2024-10-01T10:08:12.537965299-04:00" level=info msg="Loading containers: done."
Oct 01 10:08:12 osboxes dockerd[12225]: time="2024-10-01T10:08:12.635943653-04:00" level=info msg="Docker daemon" commit=41ca9
Oct 01 10:08:12 osboxes dockerd[12225]: time="2024-10-01T10:08:12.639103029-04:00" level=info msg="Daemon has completed initia
Oct 01 10:08:12 osboxes dockerd[12225]: time="2024-10-01T10:08:12.775608925-04:00" level=info msg="API listen on /run/docker.s
Oct 01 10:08:12 osboxes systemd[1]: Started Docker Application Container Engine.
~
~
~
lines 1-22/22 (END)
```

Installing Docker now gives us not just the Docker service (daemon) but also the docker command line utility, or the Docker client.

Fifth part: Docker containers provisioning

We can now provision Docker nodes.

```
osboxes@osboxes:~/Desktop$ docker info
Client:
Context:    default
Debug Mode: false
Plugins:
  app: Docker App (Docker Inc., v0.9.1-beta3)
  buildx: Docker Buildx (Docker Inc., v0.9.1-docker)
  scan: Docker Scan (Docker Inc., v0.17.0)
Server:
ERROR: Got permission denied while trying to connect to the Docker daemon socket at unix:///var/run/docker.sock: connect: permission denied
errors pretty printing info
```

- Pull an Ubuntu image:

```
osboxes@osboxes:~/Desktop$ sudo docker pull ubuntu
Using default tag: latest
latest: Pulling from library/ubuntu
dafa2b0c44d2: Pull complete
Digest: sha256:dfc10878be8d8fc9c61cbff33166cb1d1fe44391539243703c72766894fa834a
Status: Downloaded newer image for ubuntu:latest
docker.io/library/ubuntu:latest
```

- Execute Ubuntu instance image CT1:

```
osboxes@osboxes:~/Desktop$ sudo docker run --name ct1 -it ubuntu
docker: Error response from daemon: Conflict. The container name "/ct1" is already in use by container "bc306bc8711a81a8244b0206b97844ba49da025861fa7b1750ad462e01c939". You have to remove (or rename) that container to be able to reuse that name.
See 'docker run --help'.
```

- Install required connectivity testing tools:

```
osboxes@osboxes:~/Desktop$ sudo apt-get -y update && apt-get -y install net-tools iputils-ping
Hit:1 http://us.archive.ubuntu.com/ubuntu jammy InRelease
Hit:2 http://security.ubuntu.com/ubuntu jammy-security InRelease
Hit:3 http://us.archive.ubuntu.com/ubuntu jammy-updates InRelease
Hit:4 http://us.archive.ubuntu.com/ubuntu jammy-backports InRelease
Hit:5 https://download.docker.com/linux/ubuntu jammy InRelease
Reading package lists... Done
E: Could not open lock file /var/lib/dpkg/lock-frontent - open (13: Permission denied)
E: Unable to acquire the dpkg frontend lock (/var/lib/dpkg/lock-frontent), are you root?
```

- Check the connectivity:
 - o *Ifconfig* to get IP Address ➔ 172.17.0.2:

```
root@1b0d6ed3199b:/# 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 2292 bytes 25252917 (25.2 MB)
    RX errors 0 dropped 0 overruns 0 frame 0
    TX packets 1231 bytes 71363 (71.3 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 (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
```

Each Docker container use a virtual network interface (“bridge”) where they get a distinct IP address which is part of an internal subnet managed by Docker.

- Ping Internet resource from Docker was a success:

```
root@1b0d6ed3199b:/# ping google.com
PING google.com (142.250.200.238) 56(84) bytes of data.
64 bytes from mrs08s18-ln-f14.1e100.net (142.250.200.238): icmp_seq=1 ttl=112 time=7.10 ms
s
64 bytes from mrs08s18-ln-f14.1e100.net (142.250.200.238): icmp_seq=2 ttl=112 time=7.50 ms
s
64 bytes from mrs08s18-ln-f14.1e100.net (142.250.200.238): icmp_seq=3 ttl=112 time=7.44 ms
s
^C
--- google.com ping statistics ---
3 packets transmitted, 3 received, 0% packet loss, time 2005ms
rtt min/avg/max/mdev = 7.097/7.344/7.496/0.176 ms
root@1b0d6ed3199b:/#
```

The Docker container was able to contact an external resource on the internet, proving that the NAT network configured for Docker allows the container to access the outside.

- Ping VM from Docker was a success too:

```
root@1b0d6ed3199b:/# 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=0.028 ms
64 bytes from 172.17.0.2: icmp_seq=2 ttl=64 time=0.055 ms
^C
--- 172.17.0.2 ping statistics ---
2 packets transmitted, 2 received, 0% packet loss, time 1011ms
```

Docker can contact the VM hosting the container, which demonstrates that the internal communication between the two environments (Docker container and VM) works.

- Ping the Docker from the VM worked well:

```
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=0.063 ms
64 bytes from 172.17.0.2: icmp_seq=2 ttl=64 time=0.081 ms
64 bytes from 172.17.0.2: icmp_seq=3 ttl=64 time=0.081 ms
^C
--- 172.17.0.2 ping statistics ---
3 packets transmitted, 3 received, 0% packet loss, time 2052ms
rtt min/avg/max/mdev = 0.063/0.075/0.081/0.008 ms
```

Similarly, the VM was able to ping the Docker container, showing successful bi-directional connectivity between the VM and Docker.

All connectivity tests passed, showing that the network configuration between Docker, the host machine (VM), and Internet access was done correctly.

- Execute a new instance (CT2) of the ubuntu Docker:

Start a new Ubuntu container and configure port forwarding to allow SSH access from the host via port 2233

```
osboxes@osboxes:~/Desktop$ sudo docker run --name ct2 -p 2223:22 -it ubuntu
root@af5c63e5c1f3:/#
```

- Snapshot of VT2:

Saves the current state of the container as a Docker image for later reuse or sharing.

It can be useful to commit a container's file changes or settings into a new image. This lets you debug a container by running an interactive shell or export a working dataset to another server.

By default, the container being committed, and its processes will be paused while the image is committed.

```
osboxes@osboxes:~$ sudo docker commit af5c63e5c1f3 cedric:nano
sha256:12cec668724fa9e8f2371683b5f4afd6d0c4801ab20a612c5e7b80d15c986ce1
osboxes@osboxes:~$ sudo docker ps
```

CONTAINER ID	IMAGE	COMMAND	CREATED	STATUS	PORTS	NAMES
af5c63e5c1f3	ubuntu	"/bin/bash"	8 minutes ago	Up 8 minutes	0.0.0.0:2223->22/tcp	ct2

- Stop and terminate CT2:

Stops the CT2 container and deletes it to free up resources.

```
osboxes@osboxes:~$ sudo docker rm af5c63e5c1f3
af5c63e5c1f3
osboxes@osboxes:~$ sudo docker ps
```

CONTAINER ID	IMAGE	COMMAND	CREATED	STATUS	PORTS	NAMES
--------------	-------	---------	---------	--------	-------	-------

- List the available Docker images in the VM:

Shows all Docker images stored locally on the VM to see which ones are ready to use.

```
osboxes@osboxes:~$ sudo docker images
```

REPOSITORY	TAG	IMAGE ID	CREATED	SIZE
cedric	nano	12cec668724f	4 minutes ago	120MB
ubuntu	latest	b1e9cef3f297	4 weeks ago	78.1MB
ubuntu	<none>	216c552ea5ba	24 months ago	77.8MB

- Execute a new instance CT3 from the previous snapshot:

Starts a new CT3 instance based on the previous image, retaining the installed environment and tools (like Nano).

```
osboxes@osboxes:~$ sudo docker run --name ct3 -it cedric:nano
root@03b5418abf06:/#
```

- Make a proper recipe with DockerFile:

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



```
osboxes@osboxes:~/Documents$ sudo docker build -t cedric:nano -f myDocker.dockerfile .
[+] Building 24.8s (7/7) FINISHED
=> [internal] load build definition from myDocker.dockerfile
=> => transferring dockerfile: 124B
=> [internal] load metadata for docker.io/library/ubuntu:latest
=> [internal] load .dockerignore
=> => transferring context: 2B
=> [1/3] FROM docker.io/library/ubuntu:latest
=> [2/3] RUN apt update -y
=> [3/3] RUN apt install -y nano
=> exporting to image
=> => exporting layers
=> => writing image sha256:715e6fd7b3693555c9c1f376b108ee7e01a744067b61f7f5d37184b1cb489f
=> => naming to docker.io/library/cedric:nano
```

With the automated script (Dockerfile), we created a custom Docker image with project-specific dependencies and configurations.

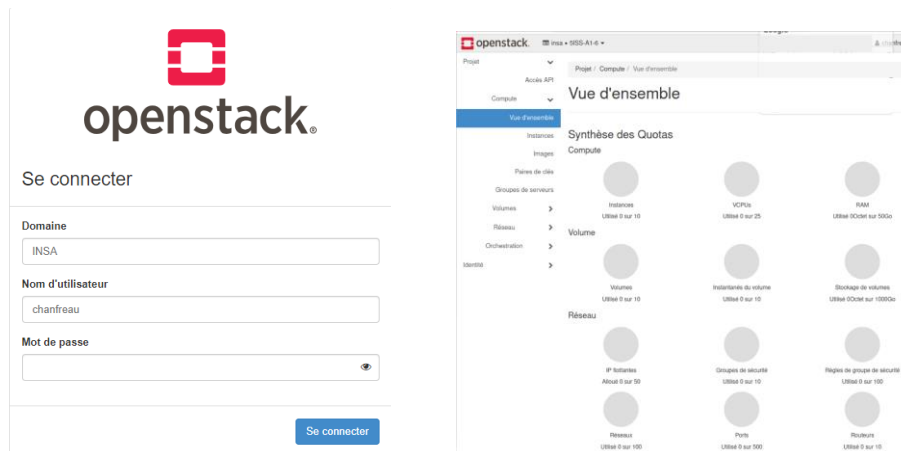
```
osboxes@osboxes:~/Desktop$ sudo docker images
```

REPOSITORY	TAG	IMAGE ID	CREATED	SIZE
cedric	nano	715e6fd7b369	9 days ago	120MB
<none>	<none>	809feb5d8c32	9 days ago	78.1MB
<none>	<none>	12cec668724f	9 days ago	120MB
ubuntu	latest	b1e9cef3f297	6 weeks ago	78.1MB
ubuntu	<none>	216c552ea5ba	2 years ago	77.8MB

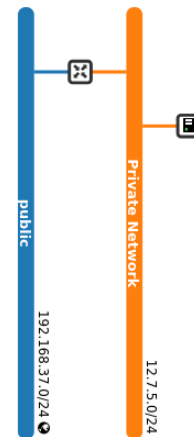
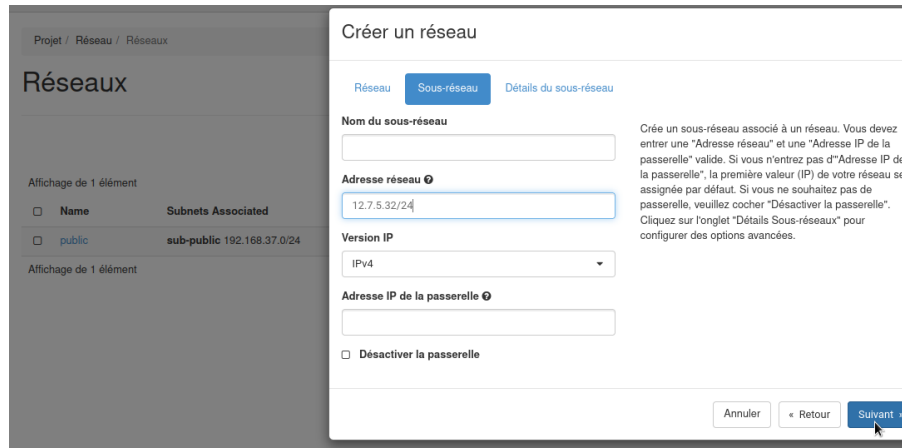
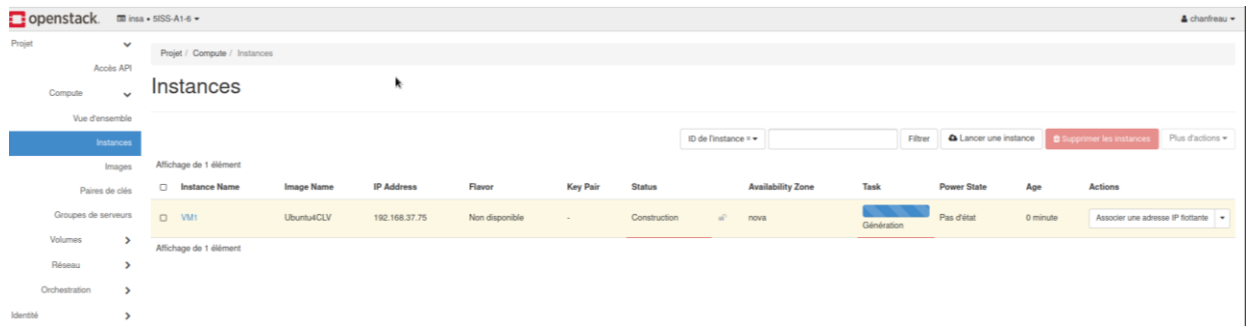
Objectives 6 and 7:

First Part: CT creation and configuration OpenStack

- OpenStack Connexion/Authentication:



- Creation of a VM / private network and gateway



A VM cannot be created from a public network for different reasons, so we created a private network:

- Network separation: The private network allows to isolate VMs from other networks. Indeed, this means that VMs can communicate with each other without being directly exposed to external threats or unauthorized traffic from the Internet.
- Traffic control: We can better manage incoming and outgoing traffic. The router plays an important role because it controls the flow between private and public network.

To allow ICMP traffic (ping) and SSH, we added three new security rules:

Gérer les règles du groupe de sécurité : default
(b4181ac3-3b34-41dd-8430-9e1d7b381ed6)

								+ Ajouter une règle Supprimer les Règles
Affichage de 7 éléments								
<input type="checkbox"/>	Direction	Ether Type	IP Protocol	Port Range	Remote IP Prefix	Remote Security Group	Description	Actions
<input type="checkbox"/>	Sortie	IPv4	Tous	Tous	0.0.0.0/0	-	-	Supprimer une Règle
<input type="checkbox"/>	Sortie	IPv4	ICMP	Tous	0.0.0.0/0	-	-	Supprimer une Règle
<input type="checkbox"/>	Sortie	IPv6	Tous	Tous	:::0	-	-	Supprimer une Règle
<input type="checkbox"/>	Entrée	IPv4	Tous	Tous	-	default	-	Supprimer une Règle
<input type="checkbox"/>	Entrée	IPv4	ICMP	Tous	0.0.0.0/0	-	-	Supprimer une Règle
<input type="checkbox"/>	Entrée	IPv4	TCP	22 (SSH)	0.0.0.0/0	-	-	Supprimer une Règle
<input type="checkbox"/>	Entrée	IPv6	Tous	Tous	-	default	-	Supprimer une Règle

Second Part: Connectivity test

First, we associated an floating IP for the VM to access the VM from the outside thanks ssh.
IP address from VM: The IP address displayed on the dashboard:

- 192.168.37.28: Public address useful to access the VM from outside.
- 12.7.5.176: Private address to communicate with another machine of the private network

<input type="checkbox"/>	Instance Name	Image Name	IP Address
<input type="checkbox"/>	VM1	Ubuntu4CLV	12.7.5.176, 192.168.37.28

Here, the router is important because it allows us to link the private network where the VM is connected and the public network.

Connectivity Test:

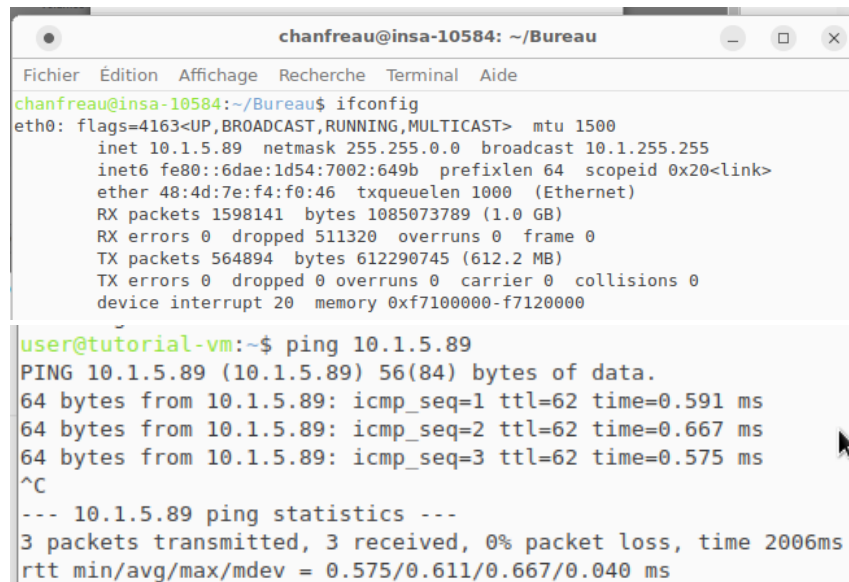
Ping Google from the VM indicates that the network configuration is properly set up, allowing the VM to communicate with external tools.

```
user@tutorial-vm:~$ ping www.google.fr
PING www.google.fr (142.251.37.35) 56(84) bytes of data.
64 bytes from mrs09s13-in-f3.1e100.net (142.251.37.35): icmp_seq=1 ttl=113 time=6.65 ms
64 bytes from mrs09s13-in-f3.1e100.net (142.251.37.35): icmp_seq=2 ttl=113 time=6.46 ms
64 bytes from mrs09s13-in-f3.1e100.net (142.251.37.35): icmp_seq=3 ttl=113 time=6.49 ms
^C
--- www.google.fr ping statistics ---
3 packets transmitted, 3 received, 0% packet loss, time 2003ms
rtt min/avg/max/mdev = 6.465/6.536/6.651/0.124 ms
```

Pinging the VM from the desktop verifies that incoming connections to the VM are working as intended. This indicates that the VM's firewall and security group settings are appropriately configured to allow traffic from the desktop.

```
chanfreau@srv-tp05:~/Bureau$ ping 192.168.37.28
PING 192.168.37.28 (192.168.37.28) 56(84) bytes of data.
64 bytes from 192.168.37.28: icmp_seq=1 ttl=61 time=2.57 ms
64 bytes from 192.168.37.28: icmp_seq=2 ttl=61 time=0.983 ms
64 bytes from 192.168.37.28: icmp_seq=3 ttl=61 time=0.906 ms
64 bytes from 192.168.37.28: icmp_seq=4 ttl=61 time=0.771 ms
^C
--- 192.168.37.28 ping statistics ---
4 packets transmitted, 4 received, 0% packet loss, time 3068ms
rtt min/avg/max/mdev = 0.771/1.306/2.567/0.731 ms
```

The ability to ping the desktop from the VM confirms that the internal network configuration is functioning correctly. This ensures that the VM can communicate with other devices on the same network.

A screenshot of a terminal window titled 'chanfreau@insa-10584: ~/Bureau'. The terminal shows the output of the 'ifconfig' command for the 'eth0' interface, displaying IP address 10.1.5.89, netmask 255.255.0.0, and other network statistics. Below this, a 'ping' command is executed from a 'user@tutorial-vm' prompt, showing three successful pings to 10.1.5.89 with varying response times. The terminal window has a menu bar with options like 'Fichier', 'Édition', 'Affichage', 'Recherche', 'Terminal', and 'Aide'.

```
chanfreau@insa-10584:~/Bureau$ ifconfig
eth0: flags=4163<UP,BROADCAST,RUNNING,MULTICAST>  mtu 1500
    inet 10.1.5.89  netmask 255.255.0.0  broadcast 10.1.255.255
    inet6 fe80::6dae:1d54:7002:649b  prefixlen 64  scopeid 0x20<link>
    ether 48:4d:7e:f4:f0:46  txqueuelen 1000  (Ethernet)
    RX packets 1598141  bytes 1085073789 (1.0 GB)
    RX errors 0  dropped 511320  overruns 0  frame 0
    TX packets 564894  bytes 612290745 (612.2 MB)
    TX errors 0  dropped 0 overruns 0  carrier 0  collisions 0
    device interrupt 20  memory 0xf7100000-f7120000

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=62 time=0.591 ms
64 bytes from 10.1.5.89: icmp_seq=2 ttl=62 time=0.667 ms
64 bytes from 10.1.5.89: icmp_seq=3 ttl=62 time=0.575 ms
^C
--- 10.1.5.89 ping statistics ---
3 packets transmitted, 3 received, 0% packet loss, time 2006ms
rtt min/avg/max/mdev = 0.575/0.611/0.667/0.040 ms
```

Network Topology:

The network topology diagram allows to visualize the different networks, routers and instances deployed in the infrastructure. This gives a clear overview of the connections and routing. By analysing this diagram, we can check how the VMs are connected to each other and to the outside.

Problem Possible:

A problem can be the absence or misconfiguration of the connection between the private network and the public network. If the instances are only connected to a private network without a gateway to the public network or the Internet, it will not be possible to access the outside or be accessible from it.

In our case, the problem could come from the absence of a correctly configured router to connect the private and public networks, or from bad routing between the subnets.

New topology creation:

- Create a new network with the gateway definition (192.168.37.1)
- Create a new VM by selecting private network
- Make sure about the coherency between private network address and router.
- Create a new router that will be the role of link between private and public network.
- Create a floating IP to be able to access the VM by the outside.
- SSH connectivity to the VM.

As SSH connectivity is working, this proves that the network configuration, router, and floating IP are configured correctly.

Third part: Snapshot, restore and resize a VM

Resize a running and a shutdown VM was not possible, and an error occurred because of the rights access.

Danger: Une erreur s'est produite. Veuillez réessayer ultérieurement. ✕

For the two previous tests, the limitation highlighted that to resize a VM, some administrative rights are necessary. A solution could be to give user role allowing some operations of resources management. This will allow more users to benefit from the flexibility of virtualization without compromising security.

By creating a snapshot of the VM, we were able to capture the current state of the system, including all files, configurations, and software installed at that time.

Therefore, if changes are made to the VM after the snapshot is created, those changes will not be present in the snapshot, which could be useful for reverting to a previous state in case of a problem.

Lancer Instance

Détails *
Source
Gabarit *
Réseaux *
Ports réseaux
Groupes de sécurité
Key Pair
Configuration
Groupes de serveurs
Scheduler Hints
Métadonnées

Les gabarits sont en place pour gérer la taille de la capacité de stockage, de mémoire et de calcul d'une instance.

Alloué

Nom	VCPUS	RAM	Total Disque	Disque Racine	Disque Éphémère	Publique
Sélectionner un élément depuis les éléments disponibles ci-dessous						

▼ Disponible 11 Sélectionnez-en une

Q Cliquez ici pour les filtres ou la recherche plein texte.

Nom	VCPUS	RAM	Total Disque	Disque Racine	Disque Éphémère	Publique
> nano	1	512 Mo	8 Go	8 Go	0 Go	Oui
> tiny	1	512 Mo	10 Go	10 Go	0 Go	Oui
> small2	1	1 Go	20 Go	20 Go	0 Go	Oui
> medium	2	2 Go	40 Go	40 Go	0 Go	Oui
> large	2	4 Go	40 Go	40 Go	0 Go	Oui
> small	1	4 Go	20 Go	20 Go	0 Go	Oui

When restoring the VM from the latest snapshot, we were able to find the exact state of the machine at the time the snapshot was created and also select new size for the VM. This allowed us to quickly recover a working system after unwanted changes or crashes

Objectives 8 & 9:

First Part: OpenStack client installation

- OpenStack client installation on Ubuntu VM.
- Configure the client with the command that loads the variables from the RC file into our current terminal session, enabling communication with the OpenStack environment.



- Start OpenStack client, this opens the OpenStack CLI.
- To get a list of available commands and options within the OpenStack CLI, we displayed the help menu by typing with *help* command.
- To get more detailed help on a specific OpenStack command, such as how to list projects, we appended project list –help.

Second Part: Web 2-tier application topology and specification

The application includes Node.js microservices to perform arithmetic operations: addition, subtraction, multiplication, and division, as well as a calculator service that handles HTTP requests.

Instances

ID de l'instance =

Filtrer

Lancer une instance

Supprimer les instances

Plus d'actions

Affichage de 6 éléments

<input type="checkbox"/>	Instance Name	Image Name	IP Address	Flavor	Key Pair	Status	Availability Zone	Task	Power State	Age	Actions
<input type="checkbox"/>	-	Ubunt u4CLV	12.7.5.110	small2	-	Active	nova	Aucun	En fonctionnement	21 minutes	Créer un instantané
<input type="checkbox"/>	Calculato	Ubunt u4CLV	12.7.5.104	small2	-	Active	nova	Aucun	En fonctionnement	22 minutes	Créer un instantané
<input type="checkbox"/>	/	Ubunt u4CLV	12.7.5.221	small2	-	Active	nova	Aucun	En fonctionnement	23 minutes	Créer un instantané
<input type="checkbox"/>	*	Ubunt u4CLV	12.7.5.138	small2	-	Active	nova	Aucun	En fonctionnement	24 minutes	Créer un instantané
<input type="checkbox"/>	+	Ubunt u4CLV	12.7.5.250	small2	-	Active	nova	Aucun	En fonctionnement	25 minutes	Créer un instantané
<input type="checkbox"/>	VM1	Ubunt u4CLV	12.7.5.176, 192.168.37.28	small2	-	Active	nova	Aucun	En fonctionnement	1 heure, 28 minutes	Créer un instantané

To deploy each microservices on the VM multiple steps was required:

- Instance connection: Here we used ssh for ease of use

```
|chanfreau@srv-tp06:~/Bureau$ ssh user@192.168.37.174 -p 22
```

- Dependencies installation: Installation of Nodejs (microservices execution), npm (Nodejs dependencies management) and curl (HTTP request).

```
user@tutorial-vm:~$ sudo apt install nodejs
user@tutorial-vm:~$ sudo apt install npm
```

- Microservices installation: Download JavaScript files inside VM of the microservice we want to deploy.

```
user@tutorial-vm:~$ wget http://homepages.laas.fr/smedjah/tmp/SumService.js
URL transformed to HTTPS due to an HSTS policy
--2024-10-08 13:11:38-- https://homepages.laas.fr/smedjah/tmp/SumService.js
Résolution de homepages.laas.fr (homepages.laas.fr)... 195.83.132.137, 2001:660:6602:2::8489
Connexion à homepages.laas.fr (homepages.laas.fr)|195.83.132.137|:443... connecté.
requête HTTP transmise, en attente de la réponse... 200 OK
Taille : 723 [application/javascript]
Enregistre : «SumService.js»

SumService.js      100%[=====]          723  --.-KB/s   ds 0s
2024-10-08 13:11:38 (55,6 MB/s) - «SumService.js» enregistré [723/723]
```

- Microservice startup: Script execution to listen request inside the specified port.

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

Listening on port : 50001
```

- Launch operations: We used curl in the VM1 to send a POST request on each service:

- o SumService

```
user@tutorial-vm:~$ curl -d "2 3" -X POST http://12.7.5.250:50001
5
```

(+)

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

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

VM1

- o Subservice:

```
user@tutorial-vm:~$ curl -d "5 3" -X POST http://12.7.5.110:50002
2
```

(-)

```
user@tutorial-vm:~$ node SubService.js
Listening on port : 50002
New request :
A = 5
B = 3
A - B = 2
```

VM1

- MulService:

```
user@tutorial-vm:~$ curl -d "12 3" -X POST http://12.7.5.138:50003
36
```

(*)

```
user@tutorial-vm:~$ node MulService.js
Listening on port : 50003
New request :
A = 12
B = 3
A * B = 36
```

VM1

- DivService:

```
user@tutorial-vm:~$ curl -d "15 5" -X POST http://12.7.5.221:50004
3
```

(/)

```
user@tutorial-vm:~$ node DivService.js
Listening on port : 50004
New request :
A = 15
B = 5
A / B = 3
```

VM1

The micro-services operations work, we can now try the Calculator Service :

- IP addresses modification in the Calculator Service code

```
user@tutorial-vm: ~
GNU nano 2.9.3 CalculatorService.js

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

const PORT = process.env.PORT || 80;

const SUM_SERVICE_IP_PORT = 'http://12.7.5.250:50001';
const SUB_SERVICE_IP_PORT = 'http://12.7.5.110:50002';
const MUL_SERVICE_IP_PORT = 'http://12.7.5.138:50003';
const DIV_SERVICE_IP_PORT = 'http://12.7.5.221:50004';
```

Port of SumService : 50001

Port of SubService : 50002

Port of MulService : 50003

Port of DivService : 50004

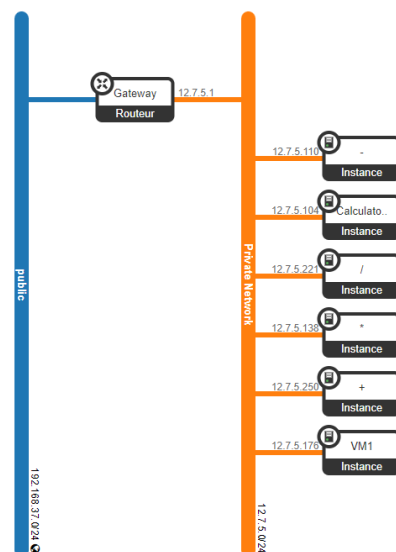
- Add the missing “sync-request” module and re start the services

```
user@tutorial-vm:~$ curl -sL https://deb.nodesource.com/setup_16.x | sudo -E bash -
user@tutorial-vm:~$ npm install sync-request
user@tutorial-vm:~$ sudo apt install nodejs
```

- o Calculator Service :

```
user@tutorial-vm:~$ curl -d "(5+6)*2" -X POST http://12.7.5.104:80
result = 22
```

```
user@tutorial-vm:~$ sudo node CalculatorService.js
Listening on port : 80
New request :
(5+6)*2 = 22
```



To communicate with the Calculator Service from the public network, we have to :

- Modify the Calculator Service port (it should be between 50000 and 50050) :

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

const PORT = process.env.PORT || 50010;

const SUM_SERVICE_IP_PORT = 'http://12.7.5.250:50001';
const SUB_SERVICE_IP_PORT = 'http://12.7.5.110:50002';
const MUL_SERVICE_IP_PORT = 'http://12.7.5.138:50003';
const DIV_SERVICE_IP_PORT = 'http://12.7.5.221:50004';
```

- To add a security rule : TCP (input/output) at port 50010 :

Ajouter une règle

Règle *

Règle TCP personnalisée

Description ?

Direction

Sortie

Ouvrir *

Port

Port ?

50010

Ajouter une règle

Règle *

Règle TCP personnalisée

Description ?

Direction

Entrée

Ouvrir *

Port

Port ?

5010|

```

user@tutorial-vm:~$ sudo node CalculatorService.js
Listening on port : 50010
New request :
(5+6)*2 = 22

```

Calculator Service

```

osboxes@osboxes:~/Desktop$ curl -d "(5+6)*2" -X POST http://192.168.37.14:50010
result = 22

```

Extern VM

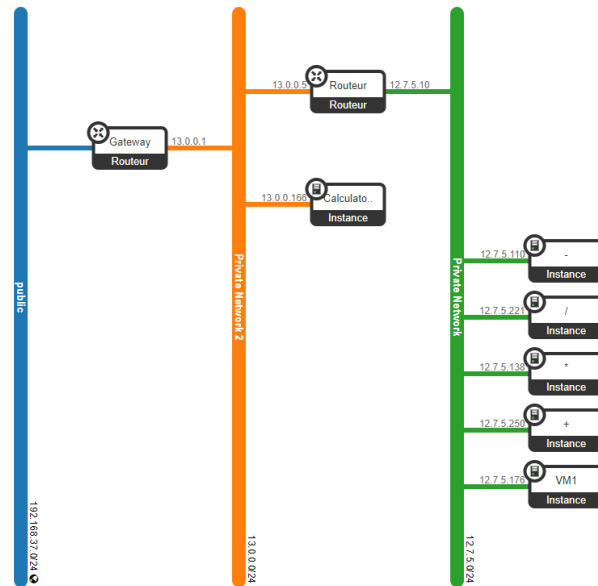
Objectives 10 and 11 :

Part one and two: The client requirements and target network topology & Deployment of the target topology

Let us consider a client that would like to deploy a Web application over a specific network topology. The latter provides secured access to intermediate services. In addition, the same client did specify that hosting VMs should be split between 2 IP networks.

For this, we have to:

- Create a new private network
- Create a new VM in the 2nd private network (Calculator Service)
- Delete the old Calculator Service
- Delete the Gateway interface to the 1st private network and add it to the 2nd private network
- Create a router between the 1st and 2nd private network
- Define a traffic route between the two private networks



- We defined the traffic route between the two private networks. It redirects the front end machine to the network 2 anytime it tries to communicate with one of the microservices.

```
user@tutorial-vm:~$ sudo route add -net 12.7.5.0/24 gw 13.0.0.5
user@tutorial-vm:~$ route
Table de routage IP du noyau
Destination      Passerelle      Genmask          Indic Metric Ref    Use Iface
default          host-13-0-0-1.l 0.0.0.0          UG    100    0      0 ens3
12.7.5.0         host-13-0-0-5.l 255.255.255.0    UG    0      0      0 ens3
13.0.0.0         0.0.0.0         255.255.255.0    U     0      0      0 ens3
169.254.169.254 13.0.0.3        255.255.255.255 UGH   100    0      0 ens3
```

- Ping calculator from net2 from a microservice/desktop ==> OK

```
PS U:\Windows\Bureau> ping 192.168.37.117

Envoi d'une requête 'Ping' 192.168.37.117 avec 32 octets de données :
Réponse de 192.168.37.117 : octets=32 temps=2 ms TTL=62
Réponse de 192.168.37.117 : octets=32 temps=1 ms TTL=62
Réponse de 192.168.37.117 : octets=32 temps=1 ms TTL=62

Statistiques Ping pour 192.168.37.117:
    Paquets : envoyés = 3, reçus = 3, perdus = 0 (perte 0%),

user@tutorial-vm:~$ ping 13.0.0.166
PING 13.0.0.166 (13.0.0.166) 56(84) bytes of data.
64 bytes from 13.0.0.166: icmp_seq=1 ttl=63 time=3.40 ms
64 bytes from 13.0.0.166: icmp_seq=2 ttl=63 time=1.22 ms
64 bytes from 13.0.0.166: icmp_seq=3 ttl=63 time=1.25 ms
^C
--- 13.0.0.166 ping statistics ---
3 packets transmitted, 3 received, 0% packet loss, time 2003ms
rtt min/avg/max/mdev = 1.227/1.961/3.404/1.020 ms
user@tutorial-vm:~$
```

Additional Part:

The goal of the exercise is to configure the microservices so that they are accessible from outside the network or from another VM located in a different private network.

- Configure Private Network 2 to allow internal connectivity between VMs, bypassing the public network.
- Configure security rules to allow internal traffic between VMs on Private Network 2.
- Ensure that the CalculatorService listens on the internal IP address of Private Network 2 (13.0.0.166) only.
- Test internal connectivity between VMs via ping and open the microservice to test it.

```
user@tutorial-vm:~$ sudo node CalculatorService.js
Listening on port : 50011
New request :
(5+6)*2 = 22
```

Calculator

```
user@tutorial-vm:~$ curl -d "(5+6)*2" -X POST http://13.0.0.166:50011
result = 22
```

VM from private network2

Automation Part:

The goal is to create a Python script capable of automatically deploying and configuring networks, instances and routers in OpenStack from a configuration file in JSON format.

- JSON file creation: it contains complete OpenStack network configuration, included network, router and instances to be deployed. This file serves as a data source for the Python script that will interpret and apply this configuration.

```
{
  "networks": [
    {
      "name": "public",
      "cidr": "192.168.37.0/24",
      "gateway": "192.168.37.254",
      "type": "external"
    },
    {
      "name": "private_network_2",
      "cidr": "13.0.0.0/24",
      "gateway": "13.0.0.1",
      "router": {
        "name": "router_1",
        "interfaces": ["13.0.0.5", "12.7.5.1"]
      }
    }
  ],
  "name": "private_network",
  "cidr": "12.7.5.0/24",
  "gateway": "12.7.5.1",
  "instances": [
    {
      "name": "Calculator",
      "ip": "13.0.0.166",
      "network": "private_network_2",
      "flavor": "small2",
      "image": "Ubuntu4CLV"
    },
    {
      "name": "instance_minus",
      "ip": "12.7.5.110",
      "flavor": "small2",
      "image": "Ubuntu4CLV"
    }
  ]
}
```

- Python Script: it is responsible for reading the JSON file and then using the OpenStack API to configure the resources described in that file.

- OpenStack Connection
- JSON reading

```
with open("network_config.json", "r") as f:  
    config = json.load(f)
```

- SubNet creation

```
for network in config["networks"]:  
    run_command(f"openstack network create {network['name']}")  
    run_command(f"openstack subnet create --network {network['name']} --subnet-range {network['cidr']} {network['name']}_subnet")
```

- Instances creation

```
for network in config["networks"]:  
    if "instances" in network:  
        for instance in network["instances"]:  
            run_command(f"openstack server create --flavor {instance['flavor']} --image {instance['image']} --nic net-id={network['name']} --fixed-ip {instance['fixed_ip']} {instance['name']}")
```

- Router creation

```
for router in config["routers"]:  
    run_command(f"openstack router create {router['name']}")  
  
    for network in router["external_network"]:  
        run_command(f"openstack router set --external-gateway {network} {router['name']}")  
  
    for network in router["internal_networks"]:  
        run_command(f"openstack router add subnet {router['name']} {network}_subnet")
```

Lab 2: Orchestrating services in hybrid cloud/edge environment

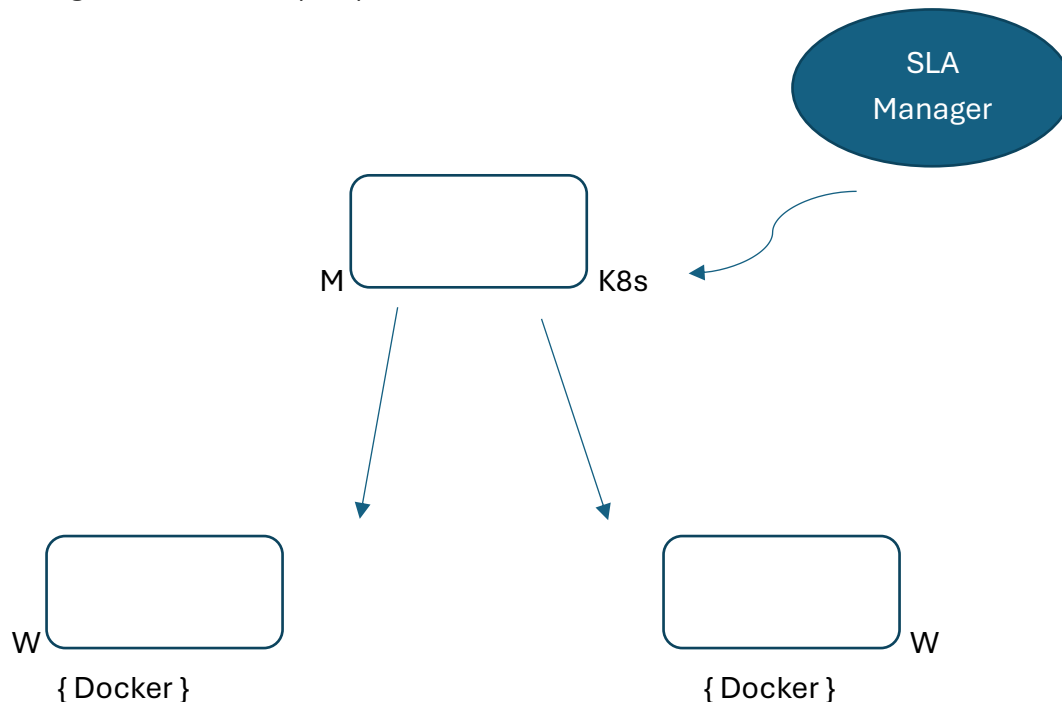
tiny.cc/TP_EdgeComputing

In this lab, we will explore implementing services in a hybrid cloud/edge environment using Kubernetes. The goal is to meet the needs of next-generation autonomous vehicles, which require real-time interactions with minimal latency to ensure optimal performance.

During this lab, we will configure a Kubernetes cluster spread across three virtual machines, simulating an intelligent infrastructure. This cluster will allow us to manage and orchestrate services deployed as Docker containers on different nodes, in order to ensure processing close to the vehicles and thus maintain very low response times.

We will also look at dynamic migration of services as the vehicle moves, thus allowing optimized resource management and adaptation to real-time needs.

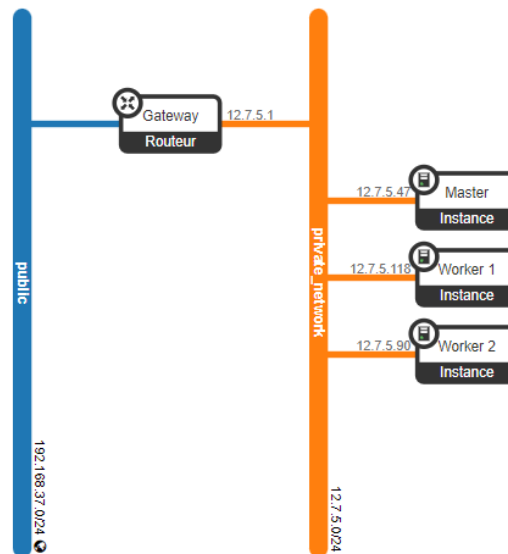
- Setup cluster K8s
 - 1- Master
 - 2- Worker nodes
- Deploy services
- Manage QoS services (SLA)



Part 1: Cloud infrastructure setup

In this part we will set up the Kubernetes cluster with 3 running nodes:

- Creation of a novel private network and connect it to the public using a well-configured gateway, instantiation of the master-node VM and 2 worker nodes VM (Ubuntu_20 cloud image and a medium flavor.):



- Generation of 3 floating IP addresses and associate them to the VM:

<input type="checkbox"/>	IP Address	Description	DNS Name	DNS Domain	Mapped Fixed IP Address	Pool	Status	Actions
<input type="checkbox"/>	192.168.37.173				Worker 1 12.7.5.118	public	Active	Dissocier ▼
<input type="checkbox"/>	192.168.37.181				Worker 2 12.7.5.90	public	Inactif	Dissocier ▼
<input type="checkbox"/>	192.168.37.78				Master 12.7.5.47	public	Active	Dissocier ▼

- On each machine some configurations were required:
 - o New user creation

```

root@ubuntu:~# adduser user
Adding user `user' ...
Adding new group `user' (1001) ...
Adding new user `user' (1001) with group `user' ...
Creating home directory `/home/user' ...
Copying files from `/etc/skel' ...
New password:
Retype new password:
passwd: password updated successfully
Changing the user information for user
Enter the new value, or press ENTER for the default
  Full Name []:
  Room Number []:
  Work Phone []:
  Home Phone []:
  Other []:
Is the information correct? [Y/n]
root@ubuntu:~#

```

- Setup the SSH connection on the VM. Open the file `sshd_config` and set `PasswordAuthentication` to “yes” using nano editor : `nano /etc/ssh/sshd_config`

```

# To disable tunneled clear text passwords, change to no here!
PasswordAuthentication yes
#PermitEmptyPasswords no

```

- Once connected with ssh on each VMs, we initialized clusters and changed host names.

```

user@ubuntu:~$ sudo swapoff -a
[sudo] password for user:
user@ubuntu:~$ sudo hostnamectl set-hostname Worker1

```

- In the next steps we updated packages, installed curl, got repository key, add Kubernetes component and marked them as hold to keep them running.
- After, we enabled docker to start, change docker group as cgroup driver before restarting it.

```

user@Worker1:~$ sudo systemctl restart docker
user@Worker1:~$ sudo docker info | grep -i cgroup
  Cgroup Driver: systemd
  Cgroup Version: 1
WARNING: No swap limit support
user@Worker1:~$ sudo modprobe overlay
user@Worker1:~$ sudo modprobe br_netfilter
user@Worker1:~$ sudo tee /etc/sysctl.d/kubernetes.conf<<EOF
> net.bridge.bridge-nf-call-ip6tables = 1
> net.bridge.bridge-nf-call-iptables = 1
> net.ipv4.ip_forward = 1
>

```

After different commands, in order to properly setup the nodes networking, we had to :

- Go to : `/usr/lib/systemd/system/kubelet.service.d`
- Open `10-kubeadm.conf` and add the flag `- node-ip` to the `KUBELET_CONFIG_ARGS` with the `IP of your master ens3 interface`.

```
user@Worker2: /usr/lib/systemd/systemd$ nano 10-kubeadm.conf
GNU nano 4.8 10-kubeadm.conf
# Note: This dropin only works with kubeadm and kubelet v1.11+
[Service]
Environment="KUBELET_KUBECONFIG_ARGS=--bootstrap-kubeconfig=/etc/kubernetes/bootstrap-kubelet.conf --kubeconfig=/etc/kubernetes/kubelet.conf"
Environment="KUBELET_CONFIG_ARGS=--config=/var/lib/kubelet/config.yaml --node-ip='12.7.5.47'"
# This is a file that "kubeadm init" and "kubeadm join" generates at runtime, populating the KUBELET_KUBEADM_ARGS variable dynamically
EnvironmentFile=/var/lib/kubelet/kubeadm-flags.env
# This is a file that the user can use for overrides of the kubelet args as a last resort. Preferably, the user should use
# the .NodeRegistration.KubeletExtraArgs object in the configuration files instead. KUBELET_EXTRA_ARGS should be sourced from this file.
EnvironmentFile=/etc/sysconfig/kubelet
ExecStart=
ExecStart=/usr/bin/kubelet $KUBELET_KUBECONFIG_ARGS $KUBELET_CONFIG_ARGS $KUBELET_KUBEADM_ARGS $KUBELET_EXTRA_ARGS
```

- Get the token to join the cluster that we copied and ran as sudo on each of the worker nodes.

```
user@Worker1:/$ sudo kubeadm join 12.7.5.47:6443 --token dcoazc.iv6
3ri9hgpyqsqil --discovery-token-ca-cert-hash sha256:a51fb1ef6f87301
a7839c558316d2775689e9b30489f0608c82593962cc1f6c5
[preflight] Running pre-flight checks
[preflight] Reading configuration from the cluster...
[preflight] FYI: You can look at this config file with 'kubectl -n
kube-system get cm kubeadm-config -o yaml'
[kubelet-start] Writing kubelet configuration to file "/var/lib/kub
elet/config.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 Bootstra
p...

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

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

- Check the status of our cluster by running the following command:

```
user@Master:/$ kubectl get nodes -o wide
```

NAME	STATUS	ROLES	AGE	VERSION	INTERNAL-IP	EXTERNAL-IP	OS-IMAGE	KERNEL-VERSION	CONTAINER-RUNTIME
master	NotReady	control-plane	5m3s	v1.28.1	12.7.5.47	<none>	Ubuntu 20.04.6 LTS	5.4.0-163-generic	containerd://1.7.12

We notice that the nodes appear when running the `kubectl get nodes -o wide` command. However, their status is "NotReady". This is due to the fact that Kubernetes itself doesn't handle networking between pods by default.

In our case we will use Calico as plug-in to ensure a proper networking. Thanks to this step, nodes will be able to communicate effectively and the cluster work as intended. To address this, we'll apply Calico on the master node using the next command, which will enable proper networking and allow the nodes to transition to the "Ready" state.


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

```
user@Master:/$ kubectl get nodes
```

NAME	STATUS	ROLES	AGE	VERSION
master	NotReady	control-plane	9m43s	v1.28.1
worker1	Ready	<none>	3m13s	v1.28.1
worker2	Ready	<none>	3m11s	v1.28.1

```
user@Master:/$ kubectl get pods -n calico
No resources found in calico namespace.
user@Master:/$ kubectl get pods -A
```

NAMESPACE	NAME	READY	STATUS	RESTARTS	AGE
kube-system	calico-kube-controllers-658d97c59c-b2zbz	1/1	Running	0	108s
kube-system	calico-node-jgsgw	1/1	Running	0	108s
kube-system	calico-node-k8rfj	1/1	Running	0	108s
kube-system	calico-node-sxp67	1/1	Running	0	108s
kube-system	coredns-5dd5756b68-q7vcs	1/1	Running	0	10m
kube-system	coredns-5dd5756b68-xv25g	1/1	Running	0	10m
kube-system	etcd-master	1/1	Running	0	10m
kube-system	kube-apiserver-master	1/1	Running	0	10m
kube-system	kube-controller-manager-master	1/1	Running	1	10m
kube-system	kube-proxy-7rb8l	1/1	Running	0	10m
kube-system	kube-proxy-cgq6x	1/1	Running	0	4m2s
kube-system	kube-proxy-xhq8d	1/1	Running	0	4m
kube-system	kube-scheduler-master	1/1	Running	1	10m

By running `kubectl get pods -A`, we get the list of pods from all namespaces in the cluster, thus displaying all active services.

Then, we will deploy ClusterIP and NodePort services to understand their usefulness in managing the network of a Kubernetes cluster.

1) Docker connection

- Docker repository connection
- Secret Docker creation: We create a secret that stores credentials to allow Kubernetes to access private Docker images in the Docker Hub repository

```
$ kubectl create secret docker-registry repo-secret --docker-  
username=<your_username> --docker-password=<your_password> --docker-  
email=<your_email>
```

- Secret verification: This command allows to verify that the secret has been created and to see the details in YAML format.

```

user@Master:/$ kubectl get secret repo-secret --output=yaml
apiVersion: v1
data:
  .dockerconfigjson: eyJhdXRocyI6eyJodHRwczovL2luZGV4LmRvY2tldi5pby92MS8iOjnsidX
  Nlcm5hbWUiOiJlc2VyIiwicGFzc3dvcmQiOiJlc2VyIiwiaXV0aCI6ImRYTmxjanAxYzJWeSJ9fX0=
kind: Secret
metadata:
  creationTimestamp: "2024-10-14T12:56:20Z"
  name: repo-secret
  namespace: default
  resourceVersion: "781"
  uid: b738cf9d-3dfc-4161-9118-292327fff21e
type: kubernetes.io/dockerconfigjson

```

2) Application deployment with ClusterIP

- Labels application on nodes: We add labels to nodes to identify and organize them. This makes it easier to select specific nodes when deploying applications in Kubernetes.

```
$ kubectl label node worker1 PoP=space_1
```

```
$ kubectl label node worker2 PoP=space_2
```

- ClusterIP service deployment: The resources defined in the ClusterIP file are deployed

```
$ kubectl apply -f ./ClusterIP
```

- Pods verification: This is to check if the pods were deployed correctly and where they are located (Pending state).

```

user@Master:/$ kubectl get pods -o wide
NAME                                READY   STATUS    RESTARTS   AGE   IP           NODE   NOMINATED NODE   READINESS GATES
fastapi-app-76f6674775-2cmh4        0/1     Pending   0           7s    <none>       <none> <none>          <none>
fastapi-app-76f6674775-655hj        0/1     Pending   0           7s    <none>       <none> <none>          <none>
fastapi-app-76f6674775-n2dcf        0/1     Pending   0           7s    <none>       <none> <none>          <none>

```

- Services verification: Display all active services in cluster.

```

user@Master:/$ kubectl get services -o wide
NAME                                TYPE        CLUSTER-IP   EXTERNAL-IP   PORT(S)    AGE   SELECTOR
fastapi-app-clusterip-service       ClusterIP    10.102.208.89 <none>        80/TCP     110s   app=fastapi-app
kubernetes                           ClusterIP    10.96.0.1     <none>        443/TCP    14m    <none>

```

```

Every 2.0s: kubectl get nodes                                Master: Mon Oct 14 13:20:12 2024

NAME        STATUS    ROLES         AGE   VERSION
master      Ready     control-plane  26m   v1.28.1
worker1     Ready     <none>         25m   v1.28.1
worker2     Ready     <none>         25m   v1.28.1

```

- Service inspection: Provides in-depth details about the service, including endpoints, which show the IP address of pods accessible through this service.

```

user@Master:~$ kubectl describe svc 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:            10.102.208.89
IPs:           10.102.208.89
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>

```

On each address on Endpoint, an application is running.

- Service connection: We send an HTTP request to one of the service endpoints to verify that the application responds correctly.

```

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

```

- Pod test creation: Pod base on Nginx image to test network access
- Pod test access: Open an interactive shell inside the pod to perform tests from it.

```
$ kubectl run testpod --image=nginx
```

```
$ kubectl exec -it testpod -- bash
```

```

user@Master:~$ kubectl exec -it testpod -- bash
Error from server: no preferred addresses found; known addresses: []

```

- Curl request : We rerun the HTTP request, but this time from inside the test pod, to check if the connectivity between pods is functional.
- Delete all resources: We remove all resources deployed for the ClusterIP service to clean up the environment before moving to the NodePort service.

```
$ kubectl delete -f ./ClusterIP
```

3) NodePort service deployment

- Deploy NodePort: This command deploys the resources defined for the NodePort service. A NodePort service exposes an application outside the cluster by assigning a port on each node in the cluster.
`$ kubectl apply -f ./NodePort`
- Pods check: Checking that pods has been deployed correctly
`$ kubectl get pods -o wide`
- Service verification: Lists the services including the NodePort and shows details like node IP and access port.
`$ kubectl get services -o wide`
- NodePort inspection: We check the details of the NodePort service, especially the port assigned by Kubernetes to access the application from outside.
`$ kubectl describe services name_of_service`
- Connectivity test: We make an HTTP request to the node IP with the assigned port to verify that the application is accessible from outside the cluster.
`$ curl http://node_1_ip:NodePort_value/`
- Delete Resources: We clean the environment by removing all resources related to the NodePort service.
`$ kubectl delete -f ./NodePort`

Conclusion

To conclude, this project set up and tested network connectivity between a virtual machine (VM), Docker containers, and the OpenStack environment. We used NAT to ensure bidirectional communication, configured port forwarding rules, and successfully tested SSH access. Duplicating VMs, provisioning Docker containers, and creating a network infrastructure in OpenStack demonstrated the flexibility of virtualized systems.