INSTITUT NATIONAL DES SCIENCES APPLIQUÉES TOULOUSE

# CLOUD AND EDGE COMPUTING

By: Biendou Brian
Brunetto Marie
Vasseur Cyril

# Table of contents

## Introduction

Cloud computing is a model for delivering computing services over the internet, providing on-demand access to resources like servers, storage, and applications. Understanding virtual machines and containers is crucial for setting up cloud computing because they enable resource consolidation, portability, isolation, flexibility, and performance optimization. Through four lessons, we learned about the differentiation and application of cloud and edge computing. This theory was completed by six lab sessions, throughout which we deepened our knowledge about virtualisation with research and application. You will see in the following sections the researches, methods, experimentations and results from these sessions.

## I - Theoretical part

Before we start working with virtualisation, we first needed to ensure that we understood the subtleties between Virtual Machines and Containerisation, as well as the differences of the technologies themselves

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

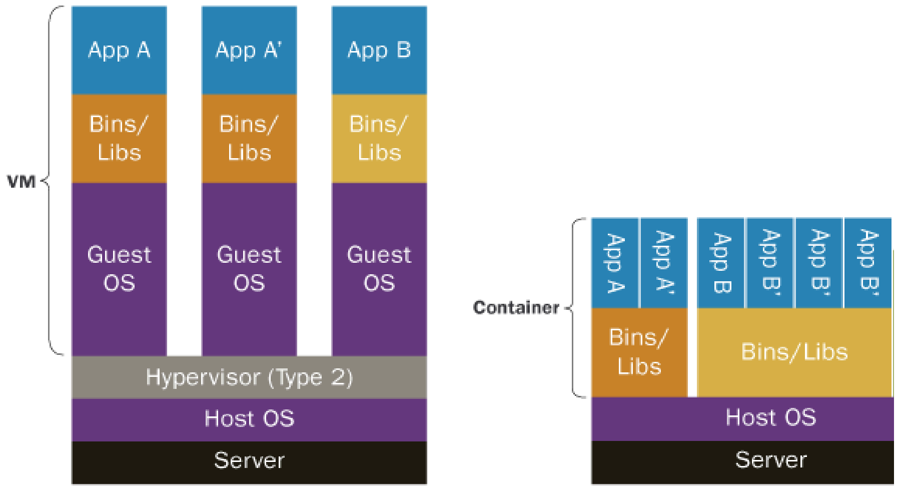


*Figure 1 - Diagram representing the difference between Virtual Machines and Containers*

The figure above illustrates the main differences between virtual machines and containers. We will compare these two technologies according to several criteria, adopting the perspectives of an application developer and an infrastructure administrator. This analysis will highlight the advantages and disadvantages of each approach in the context of their respective use.

## a. A Comparison of VMs and CTs

| | VMs | CTs |
|---|---|---|
| **From an application developer's point of view** | | |
| **Virtualization cost** | Higher cost because each VM has its own OS | Lower cost due to sharing the Host OS |
| **Usage of CPU** | Each VMs use his own resources and potentially cause an overload for application | More efficient use of resources due to the mutualisation, best performance for the application |
| **Security for the Application** | Strong isolation between VMs, reducing the risk of compromise | Less rigid security because containers share the same kernel, but they are still isolated at the process level |
| **performances** | Longer response time due to virtualization overhead | Better performance due to low virtualization overhead, |
| **Tooling** | Less flexible, tools must manage entire VMs, which are more complex to configure and manage. | More suitable for continuous integration, quick to deploy |
| **From an infrastructure administrator point of view** | | |
| **Virtualization cost** | Higher cost because each VM has its own OS | Lower cost due to sharing the Host OS |
| **Usage of CPU** | Higher utilisation as each VM consumes resources independently. | More efficient use of resources, |
| **Security for the Application** | Strong isolation between VMs, reducing the risk of compromise | Low isolation requiring other mechanism for the security |
| **performances** | Reduced performances due to overhead | Higher performance due to lightness of container |
| **Tooling** | More difficult to automate and manage | More modern and flexible |

*Table 1 - Comparison between Virtual Machines and Containers*

### b. Overall Comparison between VMs and CTs

Virtual machines (VMs) and containers (CTs) present major differences from the point of view of developers and infrastructure administrators. VMs offer better isolation and security, but at a higher cost, with performance often less optimal due to virtualization overhead. Containers, on the other hand, are lighter, quicker to deploy and better suited to modern development practices such as continuous integration, while being less costly and more efficient in their use of resources, although they offer less robust security.

## 2. Similarities and differences between the existing CT types

In addition to the difference between Virtual Machines and Container Architectures, we can notice diversity in the implementation of the containers themselves. Tools can be used to take advantage of development and features, making containerisation highly adaptable depending on the use-case.

### a. A comparison of LXC and Docker

Docker and LXC are two containerization tools that can be used on the OS Linux. Both can be applied for similar cases, while they present diversity in their features.

Docker used to rely on LXC technologies before the official 1.0 release. For that reason, they share a lot of similarities. However, for the last 10 years, Docker grew alongside LXC and developed its own strengths.
The following table summarises the comparison that can be observed between the container types from Linux and Docker:

| | **Linux LXC** | **Docker** |
|---|---|---|
| **Isolation and Resources** | Isolation is done through the use of namespaces and control groups, directly relying on the Linux kernel, providing an **excellent isolation** between the containers.[1] | Docker relies as well on namespaces and control groups. However, it is **higher-level** than Linux LXC. |
| **Containerization level** | **OS-Level Virtualization**[1] LXC containers include the applications and services, as well as the defined configuration and supporting files necessary to run them.[7] | **Application containerization**[1] Dockers proposes a more advanced containerization level. Additionally to all the features proposed by LCX, Dockers also uses different runtime for each container, improving the isolation and use of resources.[1] |
| **Tooling** | C or Python based API can be used for scripting.[2] Other tools such as continuous integration and build scripts **can be implemented**[6] | Docker uses CLI (Command-Line Interface) to manage the containers[3] It **natively** offers the use of continuous integration and allows for easier deployment[1] |

*Table 2 - Comparison between LXC and Docker based on three criteria*

## b. Overall Comparison of Containerization tools

The previous table shows that both LXC and Docker have strengths that can be challenged by other containerization tools as well, such as Podman or Containerd, CRI-O or RKT.

Each one of these tools selects to focus on specific aspects making them pertinent to choose depending on the use that is planned by the user. The following list summarise the best options depending on the chosen criteria: [8, 9, 12]

- **Isolation:** *LXC* provides the best isolation by being low-level and relying on Linux control groups and namespaces.
- **Tooling:** *Dockers* offers numerous tools facilitating workflows such as continuous integration, continuous delivery…
- **Security:** *RKT* main focus was security and reliability, and might provide the best options for this concern. However, *RKT* is not supported anymore. *Containerd* would be nowadays the best alternative, being secure by design, and doing regular vulnerability scanning, while the other tools support security features.
- **Community and support:** *Docker* is the most well-known and widely used containerization tool. This allows them to have more online resources and support from forums, making it easier for new users.
- **Orchestration:** *Containerd*, *CRI-O* and *Podman* are very well suited for orchestration systems such as Kubernetes. *LXC* and *Dockers* also provide well-thought orchestration support, while RKT never supported that feature.
- **Resource-Efficiency:** *Podman* is a lightweight and resource-efficient tool with no daemons running aside. *LXC* offers as well a good compromise between efficiency and simplicity, by running directly on the Linux Kernel.

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

**Type 1:** Hypervisor bare-metal runs directly on a machine's physical hardware, without the need for an underlying operating system. They manage hardware resources such as memory and CPU, enabling virtual machines to be created and run. This type of hypervisor is often used in data centres where performance, efficiency and isolation are crucial.

- ❖ High security: By completely isolating each virtual machine, Type 1 guarantees strong protection against vulnerabilities.
- ❖ High performance with low latency: Bypassing a host OS enables near-native performance, as there are no intermediate layers to manage.

**Type 2 :** hypervisor Hosted runs on top of a pre-existing operating system. They rely on this OS to access hardware resources. This type of hypervisor is generally simpler to use and configure, making it ideal for test, development or small-scale virtualization scenarios.

- ❖ Easy to use and configure: Its rapid installation and flexibility make it a popular choice for developers.
- ❖ Less secure and less powerful than Type 1, as it inherits the vulnerabilities and limitations of the host operating system.

VirtualBox is a type 2 hypervisor. It runs on top of an OS such as Windows or Linux OpenStack is a type 1 hypervisor and complete cloud platform making it suitable for large-scale production environments.

When comparing Type 1 and Type 2 hypervisors, it's clear that they address different virtualization needs. Type 1 hypervisors offer optimum performance and enhanced security, making them ideal for demanding production environments. On the other hand, Type 2 hypervisors, with their ease of use and flexibility, are better suited to developers and test scenarios. The main difference lies in the underlying architecture, where type 1 runs directly on hardware, while type 2 depends on a host operating system, resulting in significant differences in terms of performance and security. So, the choice between these two types of hypervisor must be made according to the specific requirements of the operating environment and priorities in terms of security and performance.

# II - Practical part

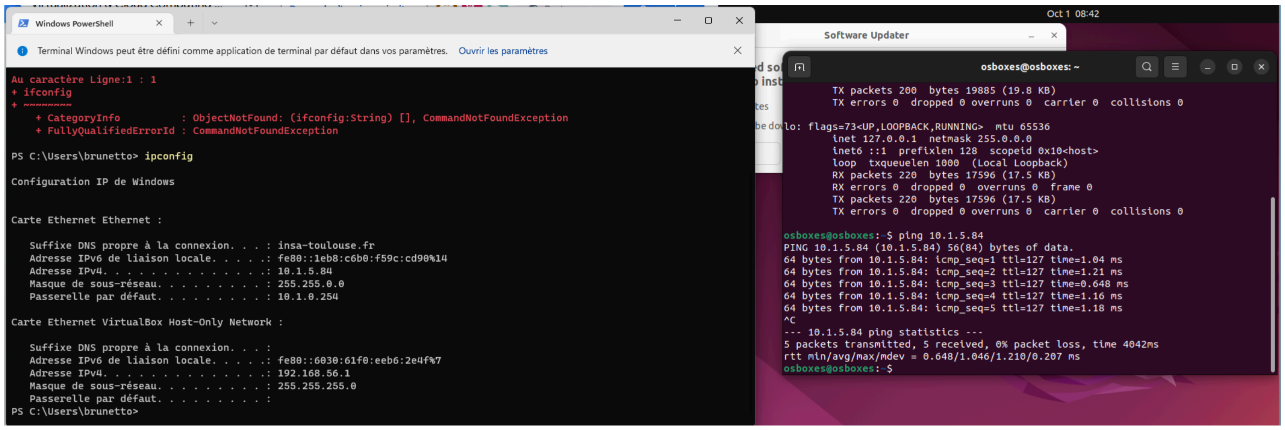
Once our comprehension of the virtual machines and Containers has been deepened, it was time to apply it to more concrete examples. Through the practical part, we will present the test we conducted to validate our knowledge of those concepts, as well as the basic functionalities offered by cloud service providers.

## 1. Getting familiar with virtualization

To explore the functionalities of virtualization, we first needed to get familiar with virtualization tools. For this purpose, we used VirtualBox for Virtual Machines and Docker for Containerisation.

### a. VirtualBox for Virtual Machines

VirtualBox is a hypervisor created by Oracle for visualisation purposes. This is the software we will use for the following tests.
Once we successfully launched our virtual machine, the first test was to ensure the connectivity of it with other hosts/ the internet.


Windows PowerShell
Terminal Windows peut être défini comme application de terminal par défaut dans vos paramètres. Ouvrir les paramètres
Oct 1 08:42
Software Updater
osboxes@osboxes: ~

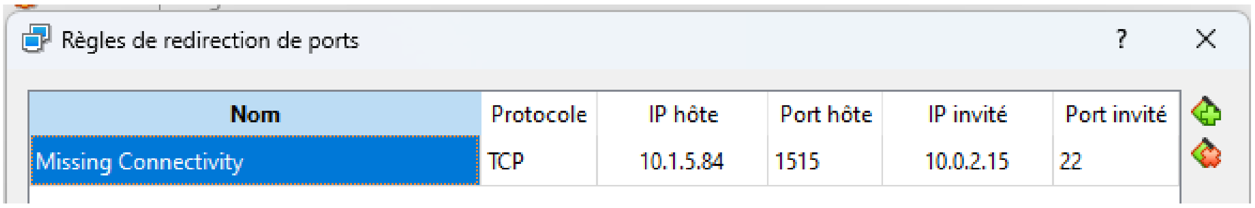
```
Au caractère Ligne:1 : 1
+ ifconfig
+ ~~~~~~~~
    + CategoryInfo          : ObjectNotFound: (ifconfig:String) [], CommandNotFoundException
    + FullyQualifiedErrorId : CommandNotFoundException

PS C:\Users\brunetto> ipconfig

Configuration IP de Windows


Carte Ethernet Ethernet :

   Suffixe DNS propre à la connexion. . . : insa-toulouse.fr
   Adresse IPv6 de liaison locale. . . . .: fe80::1eb8:c6b0:f59c:cd90%14
   Adresse IPv4. . . . . . . . . . . . . .: 10.1.5.84
   Masque de sous-réseau. . . . . . . . . : 255.255.0.0
   Passerelle par défaut. . . . . . . . . : 10.1.0.254

Carte Ethernet VirtualBox Host-Only Network :

   Suffixe DNS propre à la connexion. . . :
   Adresse IPv6 de liaison locale. . . . .: fe80::6030:61f0:eeb6:2e4f%7
   Adresse IPv4. . . . . . . . . . . . . .: 192.168.56.1
   Masque de sous-réseau. . . . . . . . . : 255.255.255.0
   Passerelle par défaut. . . . . . . . . :
PS C:\Users\brunetto>
```

```
        TX packets 200  bytes 19885 (19.8 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 220  bytes 17596 (17.5 KB)
        RX errors 0  dropped 0  overruns 0  frame 0
        TX packets 220  bytes 17596 (17.5 KB)
        TX errors 0  dropped 0 overruns 0  carrier 0  collisions 0

osboxes@osboxes:~$ ping 10.1.5.84
PING 10.1.5.84 (10.1.5.84) 56(84) bytes of data.
64 bytes from 10.1.5.84: icmp_seq=1 ttl=127 time=1.04 ms
64 bytes from 10.1.5.84: icmp_seq=2 ttl=127 time=1.21 ms
64 bytes from 10.1.5.84: icmp_seq=3 ttl=127 time=0.648 ms
64 bytes from 10.1.5.84: icmp_seq=4 ttl=127 time=1.16 ms
64 bytes from 10.1.5.84: icmp_seq=5 ttl=127 time=1.18 ms
^C
--- 10.1.5.84 ping statistics ---
5 packets transmitted, 5 received, 0% packet loss, time 4042ms
rtt min/avg/max/mdev = 0.648/1.046/1.210/0.207 ms
osboxes@osboxes:~$
```

*Figure 2 - Successful ping to test connectivity from the Virtual Machine to the host*

Currently, the connectivity test with the outside hosts failed due to the one-way NAT. This means that pings can be sent and received back. However, the Virtual Machine cannot be pinged from another host as the address is not routable.
The solution was to add a rule to the NAT to forward packets addressed to the NAT with specific sending to the corresponding ports in the Virtual Machine.

In our test configuration, we dictated that all packets from the host computer (10.1.5.84) with an upcoming port of 1514 would be addressed to the virtual machine (10.0.2.15) at port 22. This allowed us to create an SSH connection that would be used to validate our theory and to be used later to set up the VM outside of the VirtualBox interface.
We would then be able to set up any other receiving port to be able to have a more complex communication with other hosts.


Règles de redirection de ports

| Nom | Protocole | IP hôte | Port hôte | IP invité | Port invité |
|---|---|---|---|---|---|
| Missing Connectivity | TCP | 10.1.5.84 | 1515 | 10.0.2.15 | 22 |

*Figure 3 - Rule for our SSH connection*

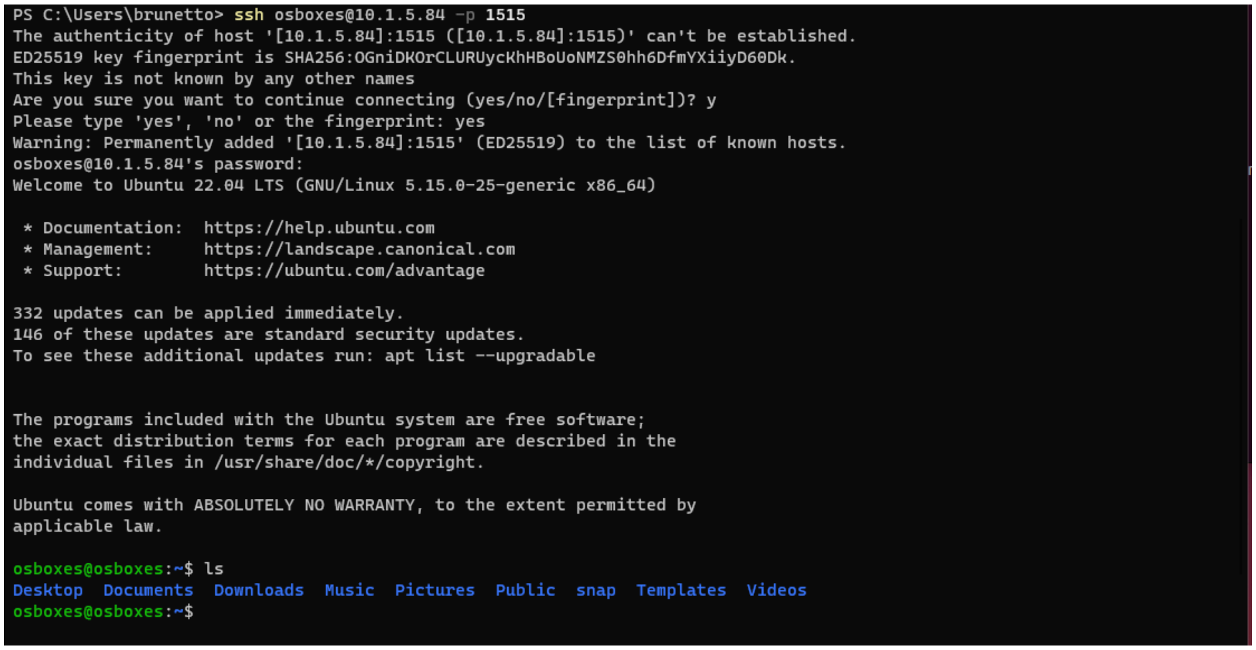

```
PS C:\Users\brunetto> ssh osboxes@10.1.5.84 -p 1515
The authenticity of host '[10.1.5.84]:1515 ([10.1.5.84]:1515)' can't be established.
ED25519 key fingerprint is SHA256:OGniDKOrCLURUycKhHBoUoNMZS0hh6DfmYXiiyD60Dk.
This key is not known by any other names
Are you sure you want to continue connecting (yes/no/[fingerprint])? y
Please type 'yes', 'no' or the fingerprint: yes
Warning: Permanently added '[10.1.5.84]:1515' (ED25519) to the list of known hosts.
osboxes@10.1.5.84'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


The programs included with the Ubuntu system are free software;
the exact distribution terms for each program are described in the
individual files in /usr/share/doc/*/copyright.

Ubuntu comes with ABSOLUTELY NO WARRANTY, to the extent permitted by
applicable law.

osboxes@osboxes:~$ ls
Desktop  Documents  Downloads  Music  Pictures  Public  snap  Templates  Videos
osboxes@osboxes:~$
```

*Figure 4 - The SSH connection from the host computer to the Virtual Machine*

Finally, we managed to use the *clonemedium* command to copy the disk file. We were able to see when we runned another Virtual Machine from the copied disk file that OpenSSH was installed, meaning that the configuration was kept during the copy.

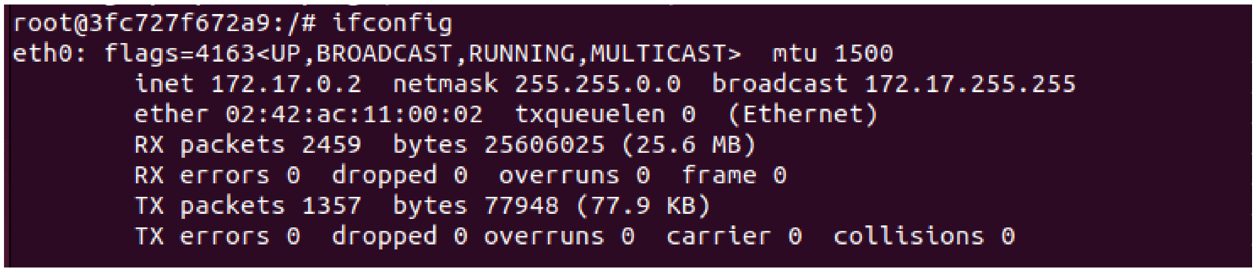
This feature can be useful when it comes to create backup or snapshot of configurations, for saving, sharing or testing purposes.

### b. Docker for Containers

Docker is an open-source platform that enables the creation, deployment, and running of applications within containers. A container is a lightweight, portable unit that bundles an application with all its dependencies (libraries, configurations, etc.) to ensure it runs consistently across different environments.

Docker requires us to use super-user commands that we do not have on INSA's machines. Therefore, we will run Docker inside a virtual machine.

We used ssh to be able to execute the command to install Docker from the host machine directly.



```
root@3fc727f672a9:/# 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 2459  bytes 25606025 (25.6 MB)
        RX errors 0  dropped 0  overruns 0  frame 0
        TX packets 1357  bytes 77948 (77.9 KB)
        TX errors 0  dropped 0 overruns 0  carrier 0  collisions 0
```

*Figure 5 - SSH on a docker instance, allowing us to see the ip and oher configs*

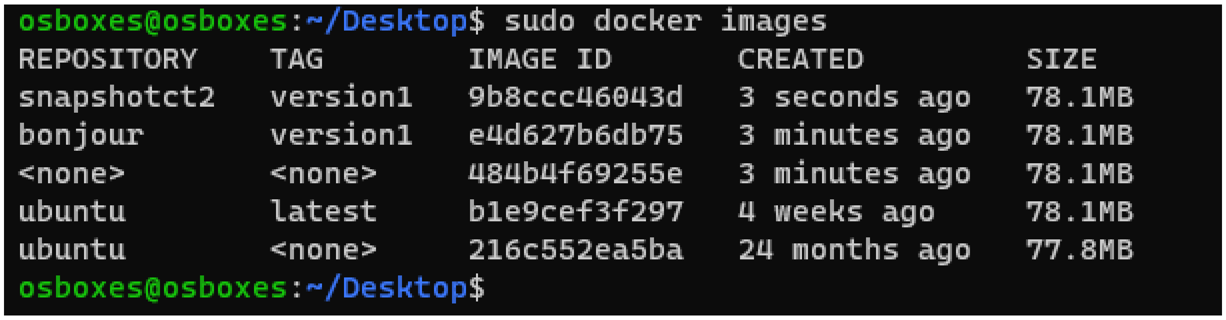
Once our container was set up, we were able to test the connectivity from the instance to compare with the ones from the virtual machines. We can there see that we were able to communicate the outside from the inside of the container through the following ping, where 10.0.2.15 is the ip address of the virtual machine on which the container exists:

```
$ ping www.google.com
```

```
$ ping 10.0.2.15
```

Both pings were successful. An additional ping from the Virtual Machine to the docker instance shows as well that it is also possible to access the container from outside, unless virtual machines. This can be explained by the fact that Docker itself manages the routing part of the containers. It is however possible to set up specific networks by a user if needed.[14]

The next experiment we had was to try to create snapshots as we did for the virtual machines. We were able to see that installed resources, here nano, were still present after the creation of an instance from the newly made snapshot.



```
osboxes@osboxes:~/Desktop$ sudo docker images
REPOSITORY     TAG        IMAGE ID       CREATED          SIZE
snapshotct2    version1   9b8ccc46043d   3 seconds ago    78.1MB
bonjour        version1   e4d627b6db75   3 minutes ago    78.1MB
<none>         <none>     484b4f69255e   3 minutes ago    78.1MB
ubuntu         latest     b1e9cef3f297   4 weeks ago      78.1MB
ubuntu         <none>     216c552ea5ba   24 months ago    77.8MB
osboxes@osboxes:~/Desktop$
```

*Figure 6 - Available images, some downloaded (both ubuntu), some snapshots (snapshotct2 and bonjour)*

## 2. Application for Distributed Services

Now that we are aware of the parameters and functionalities offered by virtual machines and containers, we are able to develop distributed systems.

To achieve this goal, we had the opportunity to use OpenStack. OpenStack is a cloud computing platform, allowing us to easily create and manage virtual machines, as well as virtual networks they might be working on.

After some discovery tests, we used the software to understand the interface and virtualization of the network. We started with the first application: a calculator system.

The aim of this application is to be able to call a function with a string which is composed of a concatenation of simple operations (addition, subtraction, division and multiplication), returning the result of the whole given formula.

Each type of operation is represented as a separated service, located on its own virtual machine.

To create the application, we first needed to make a private network on which the services would be installed. OpenStack allows the developer to manage virtual networks, as well as virtual components to link them together (router and other gateways)

In the example below, you will be able to see our implementation. The public network 192.168.37.0/24 is the network owned by INSA. The private network 12.7.5.0/24 is our newly created private network.

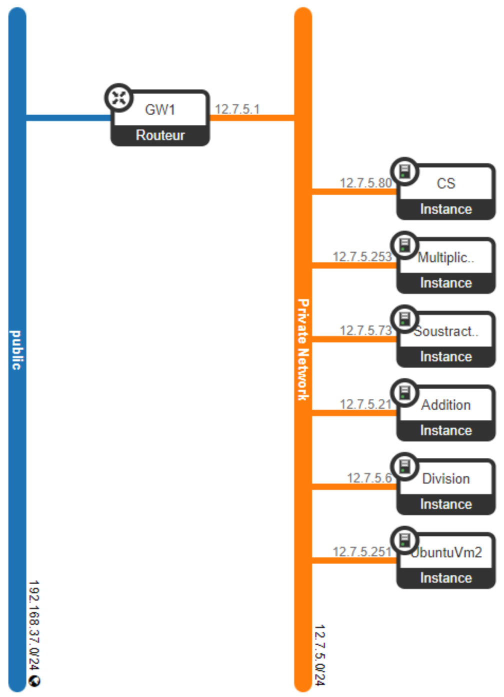


*Figure 7: Topology of our network*

A router GW1 has been added, with interfaces on both networks, to allow the communication between them, and giving access to the private network from and to the outside. However, some security rules needed to be added in order to send packets:

- ICMP to test connectivity through pings
- SSH to take control of the Virtual Machine from the host during the development

Then, we needed to create virtual machines. To achieve this, we decided to create a snapshot with the common packages (npm, curl, nodejs and apk) installed as well as some configuration, such as the AZERTY keyboard. This method would save us time. We first created a snapshot called “microservice_snapshot”. 5 Instances of this snapshot are then instantiated: Addition, Soustraction, Multiplication, Division and CS.

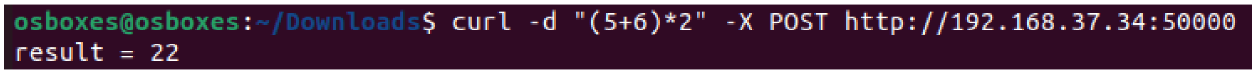
Once these virtual machines were created, we needed to individually set up each one to meet their requirements. These requirements were to download and run a nodejs application to create an interface with the appropriate calculation method.
The commands to run the service are:

```
$ wget <given path to the js file to download>
$ node <downloaded js file to run>
```

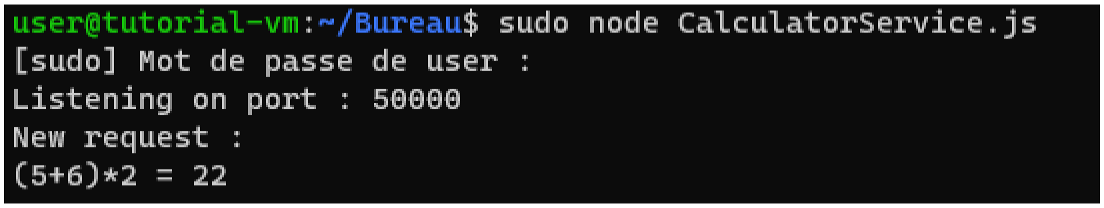
For instance, for the sum service, it would be:

```
$ wget http://homepages.laas.fr/smedjiah/tmp/SumService.js
$ node SumService.js
```

We decided to set up the virtual machines using SSH, facilitating the recuperation of commands and avoiding the problem of different keyboards per machine. However, we used a greedy approach, consisting in the creation and attribution of a floating address for each service. A better way to proceed would have been to attribute a single floating address to the gateway, allowing us to create a NAT, where the ssh would have been possible depending on the chosen port.



```
osboxes@osboxes:~/Downloads$ curl -d "(5+6)*2" -X POST http://192.168.37.34:50000
result = 22
```



```
user@tutorial-vm:~/Bureau$ sudo node CalculatorService.js
[sudo] Mot de passe de user :
Listening on port : 50000
New request :
(5+6)*2 = 22
```

*Figure 8 & 9: Result of our call to Calculator service.js from the point of view of both communicating machines*

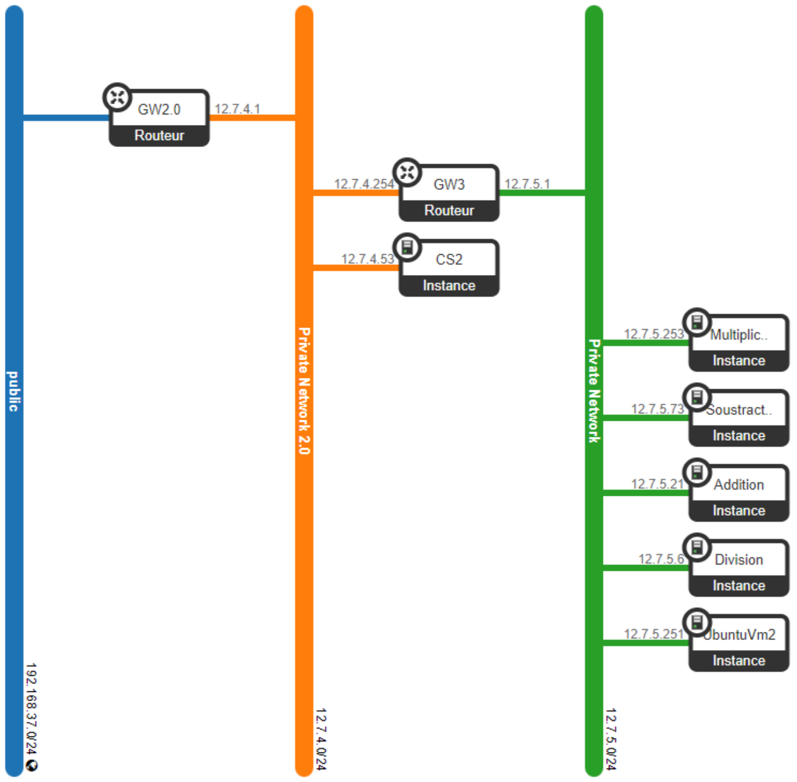
Finally, we were able to call the service on CalculatorService.js with the command:

```
$ curl -d "(5+6)*2" -X POST http://192.168.37.34:5000
```

This command gives to the service the operation (5+6)*2. The service is listening on port 5000 of the floating IP 192.168.37.34, accessible from the client machine.
We can see here that we got the right result, showing the ability of the service to call the other services (here addition and multiplication), themselves listening on the same private network.

## 3. Division of the private networks

The next step of our operation was to try to set our client on a separated private network. Openstack network management systems let the machines statically have the address of the gateway as default, predefined in the network at its creation on Openstack.



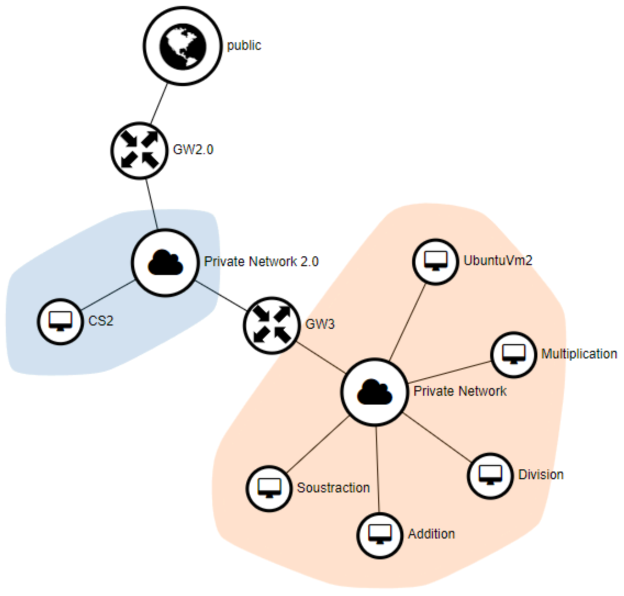


*Figure 10 & 11: Topology of the final application using two private networks*

## 4. Microservice with Dockerfile

In the previous section, we were able to experiment with services, using virtual machines. We now want to explore the same structure using containers. For that, we will be using Docker. This will also be an application for the concept we previously saw.

As we did with the snapshot for the virtual machine, we decided to factor the build by using a Dockerfile. A Dockerfile is a file containing commands and instructions used to build a Docker Container with a specific configuration.
For this Dockerfile, we made sure to download all the necessary packages (curl, wget, nodeJS and npm) as well as all the services scripts downloaded through the wget command. Every service was then an instance of this Dockerfile built, on which we execute the appropriate service.

First, to build a Dockerfile in order to have a working image, we used the following command :

```
$ sudo docker build .
```

Tree structure of our files:

```
docker_exo
       |_compose.yml
       |_microservice
             |_Dockerfile
             |_package.json
```

The first command will then create an instance of a service from the image, while the second one execute the command to run the right service:

```
$ sudo docker run -d -it <ImageName>
$ sudo docker exec -it <id> sh -c "node <ServiceName>.js"
```

```
$ sudo docker-compose up
```

The following step was to find the ip address of each service to be able to set them later in the Calculator Service. This will give the possibility to the services to be called according to the address and predefined port.
To find the ip address of the docker container, we used the following command:

```
$ sudo docker ps
$ sudo docker inspect <id>
```

And then set up in the CalculatorService.js file through the following line, executing a command allowing us to edit the file in the docker:

```
$ sudo docker exec -it <id> sh -c "nano CalculatorService.js"
```

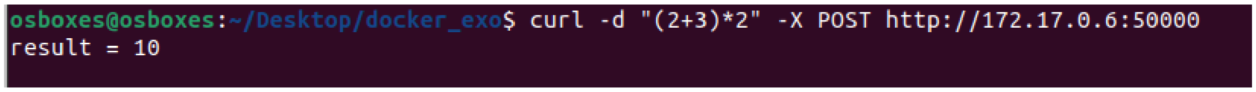
As we previously saw, networks are already managed by Docker, and thus no more configurations are needed for the container to be able to communicate.

Finally,  we runned the Calculator Service. However, in our Dockerfile, we forgot to install sync-request, which is used by this service. This package not being able to be install in the root, we created a folder in which we were able to properly run the service

```
$ sudo docker exec -it <id> sh -c "curl -sL
https://deb.nodesource.com/setup_16.x | sudo -E bash -"
$ sudo docker exec -it <id> sh -c "mkdir install_file"
$ sudo docker exec -it <id> sh -c "mv CalculatorService.js ./install_file"
$ sudo docker exec -it <id> sh -c "cd install_file && npm install
sync-request"
$ sudo docker exec -it <id> sh -c "cd install_file && node Calculator.js"
```

We could then test the calculator through the following command, with 172.17.0.6 being the ip of our Calculator Service:

```
$ curl -d "(3+2)*2" -X POST http://172.17.0.6:50000
result = 10
```



```
osboxes@osboxes:~/Desktop/docker_exo$ curl -d "(2+3)*2" -X POST http://172.17.0.6:50000
result = 10
```

*Figure 12: Result from the Calculator Service using containers*

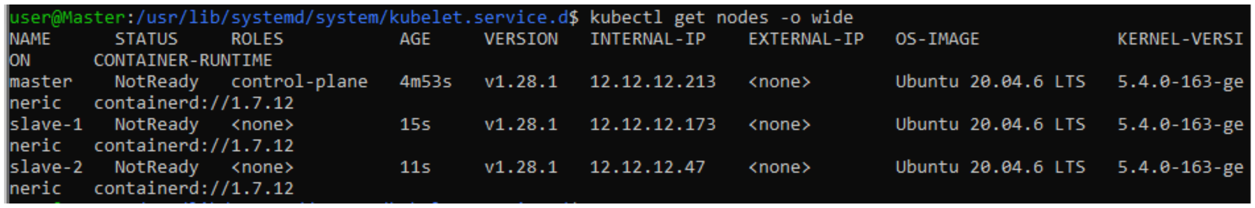
An improvement that we could have done would be to create a bash script allowing us to execute the commands launching and running each service. However, for them to work, they require their own terminal, as they cannot run as background tasks.

## III - Edge computing

This third section covers the last lab of this course. This lab was an opportunity for us to deepen our knowledge about edge technology through a theoretical application of a connected car, gathering data from its environment. The car should be able to use edge technology to run some critical services on devices with better computing power. These dynamic services would help the data to be processed faster and thus to "help sharpen decision taking".

The lab consisted in exploring the first aspect of this application: the setup of the cloud architecture surrounding the system as an introduction. To achieve this goal, we will use the orchestrator Kubernetes, on which Docker containers will be deployed to emulate the services. Finally, Openstack virtual machines will be used as hosting nodes for the services on a virtual network.

Three nodes, one master and two workers/slaves are created and set up to be able to support Kubernetes and Docker for the services to be hosted.



```
user@Master:/usr/lib/systemd/system/kubelet.service.d$ kubectl get nodes -o wide
NAME      STATUS     ROLES           AGE     VERSION   INTERNAL-IP     EXTERNAL-IP   OS-IMAGE             KERNEL-VERSI
ON      CONTAINER-RUNTIME
master    NotReady   control-plane   4m53s   v1.28.1   12.12.12.213    <none>        Ubuntu 20.04.6 LTS   5.4.0-163-ge
neric   containerd://1.7.12
slave-1   NotReady   <none>          15s     v1.28.1   12.12.12.173    <none>        Ubuntu 20.04.6 LTS   5.4.0-163-ge
neric   containerd://1.7.12
slave-2   NotReady   <none>          11s     v1.28.1   12.12.12.47     <none>        Ubuntu 20.04.6 LTS   5.4.0-163-ge
neric   containerd://1.7.12
```

*Figure 13: Overview of the nodes on kubernetes*

However, at this point, our group runned into issues regarding the installation of Kubernetes nodes, due to some unspotted typographical errors in the configuration file, on the virtual machines. The majority of this last lab was lost searching for the error. This part has then mainly been seen theoretically, and further theory and practice would be needed for us to fully understand the practical use cases of Edge applications and Kubernetes orchestrations.

## Conclusion

Through these labs, we were able to see the theory of Virtual Machines and Containers using Openstack and Docker. We had the opportunity to apply this new knowledge through exercises and an application using micro services.
These labs gave us the tools to knowingly choose when virtualization could be needed, which technology is the best to choose and even how to use some of these tools.
We would like to thank you for reading our report and we would also like to thank our TP supervisor for monitoring our progress and supporting us throughout the TP.

## References

[1] J. Perlow. (2024, Jun, 13). *LXC vs. Docker: Which One Should You Use?* [Online]. Available: https://www.docker.com/blog/lxc-vs-docker/

[2] Linux Containers. *LXC - Documentation* [Online]. Available: https://linuxcontainers.org/lxc/documentation/

[3] E. Kahuha. (2023, Jul, 11). *LXC vs Docker: Which Container Platform Is Right for You?* [Online]. Available: https://earthly.dev/blog/lxc-vs-docker/

[4] IBM Cloud Team. *Containers Versus Virtual Machines (VMs): What's The Difference?* [Online]. Available: https://www.ibm.com/think/topics/containers-vs-vms

[5] (2023, Mar, 31). *Containers vs. virtual machines* [Online]. Available: https://learn.microsoft.com/en-us/virtualization/windowscontainers/about/containers-vs-vm

[6] *GitHub - lxc/lxc-ci: LXC continuous integration and build script* [Online]. Available: https://github.com/lxc/lxc-ci

[7] (2022, May 11). *What's a Linux Container* [Online]. Available: https://www.redhat.com/en/topics/containers/whats-a-linux-container

[8] Y. Rehman. (2023, Jan 30). *Comparison of container runtimes or management technologies [Docker, containerd, Podman, rkt]* [Online]. Available: https://dev.to/theyasirr/comparison-of-container-runtimes-or-managment-technologies-docker-containerd-podman-rkt-1b8b

[9] F. Sager. (2019, Oct 22). *Docker vs LXC vs rkt* [Online]. Available: https://stackshare.io/stackups/docker-vs-lxc-vs-rkt

[10] *Quelle est la différence entre les hyperviseurs de type 1 et les hyperviseurs de type 2 ?* [Online]. Available: https://aws.amazon.com/fr/compare/the-difference-between-type-1-and-type-2-hypervisors/

[11] S. Bigelow. (2024, Mar 7). *What's the difference between Type 1 vs. Type 2 hypervisor?* [Online]. Available: https://www.techtarget.com/searchitoperations/tip/Whats-the-difference-between-Type-1-vs-Type-2-hypervisor

[12] N. Ehrman. (2024, Feb 12). *Container Runtimes Explained* [Online]. Available: https://www.wiz.io/academy/container-runtimes

[13] BBC News. (2013, Nov. 11). *Microwave signals turned into electrical power* [Online]. Available: http://www.bbc.co.uk/news/technology-24897584

[14] *Dockers Docs - Networking Overview* [Online]. Available: https://docs.docker.com/engine/network/