Cloud Computing Report

Baclé Lucas - Géhin Clément - Kassé Marc

October 2020

1 Theoretical Part

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

VMs allow multiple OS to run on top of the host computer's one. To allow several OS to run on the same computer, VMs require an hypervisor knowing how to share the hardware resources. It provides system isolation between each VMs but still relies on the same hardware.

CTs share the same OS as the host computer. It means that different CTs will for instance share the system calls and kernel functions. CTs are much lighter in computer resources (cpu / ram) because they do not run another OS on top of the host computer one, and it does not require the hypervisor. They also share binaries and libraries which further reduce the amount of resources used per application. It provides isolation between each CTs' processes but relies on the same OS.

An application developer will enjoy CTs for the easier CI CD support it allows with tools like Kubernetes. With lighter overhead per application, the response time will be shorter than VMs. But he will generally prefer VMs to test and develop applications on several systems.

An infrastructure administrator will enjoy the lighter overhead of CTs, it permits a better usage of servers resources. This way he will be able to create more isolated spaces on the same computer. But he will prefer VM for the better isolation and enhanced security it provides, if a VM OS crashes, only this VM will become down but since CTs share the same OS, the whole system could crash.

1.2 Similarities and differences between the existing CT types

CT types differ on the containerization level, for example LXC is positioned on a OS level when Docker is on an application level. What does that mean? It means that on docker each container is for one application environment. In opposition with LXC you will not be able to have access to services, daemons, syslog, cron or running multiple applications. LXC behaves as a lightweight

VMs. It allows you to log into your container , treat it like an OS and install your application and services.

LXC is like a VM with a fully functional OS when Docker uses read only layers to work. Docker uses AUFS/device mapper/btrfs to build containers using read only layers of file systems. We talk about Docker image, where an image is a container composed of layers used to build your application. Docker will also stock the data outside the container in the host since the layers are read only. This might be a limit to the portability of a container, however the read only aspect will facilitate the distribution of the application.

The fact that Docker uses containers on an application level may cause difficulties that do not exist with LXC. Today applications and services are designed to run on multi process OS. Since LXCis a functional OS you will be able to host an application in one of its containers easily. In the opposite it might be harder for Docker containers, for example if you develop an application for a LAMP container you will need to build three containers (php, Apache, MySQL) and make them interact with each other. Another drawback of Docker is the performance of the application running in the container. The multi layer design of Docker may drop the performance of your application.

From a multi-tenancy point of view, Docker and LXC have their advantages and drawbacks, it depends on what you want to do with your application. If you want to grant access to your application to different users completely isolated, the fact that the resources are stored on the host and that the application is read only will make you choose Docker. On the other hand if you want to share resources between users LXC, which runs as an OS, will be your choice.

1.3 Similarities and differences between Type 1 and Type 2 of hypervisors' architectures

In both types, each VM is situated above a hypervisor. The difference between the two types is in the position of this hypervisor. In the type 1 called Bare-Metal the hypervisor is situated just above the hardware. This means that the physical machine does not have any OS, it only hosts the VMs. In the type 2, called hosted, the hypervisor is situated above the machine (physical) OS. This means that the physical machine has its own processes alongside the different VMs.

Both Openstack and Virtualbox belong to the type 2 of architecture. Indeed, their hypervisors are installed on an existing OS (Windows most of the time). So the physical machine will continue to work normally and also host the VMs.

2 VirtualBox and Docker

2.1 Creating and configuring a VM

The host machine has two IP addresses, 172.20.10.2 which is the address of the machine on its private network and 192.168.56.1 which is the address in the network of VMs. The VM has 10.0.2.15 as an IP address, which means that it is connected to the private network 10.0.2.0/24.

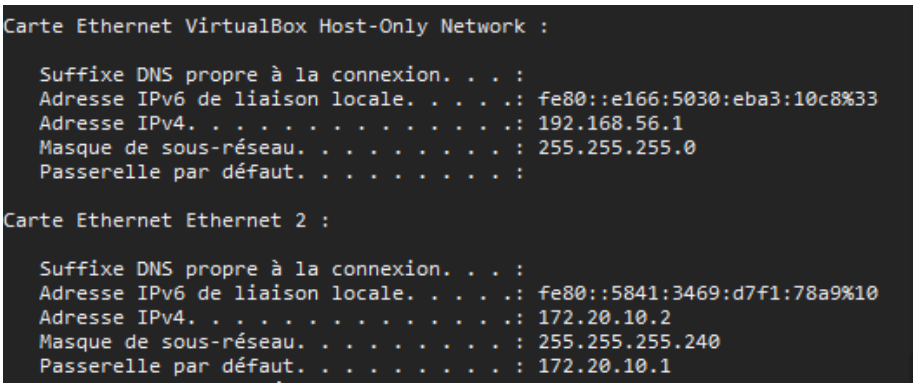

Figure 1: Host network card configuration

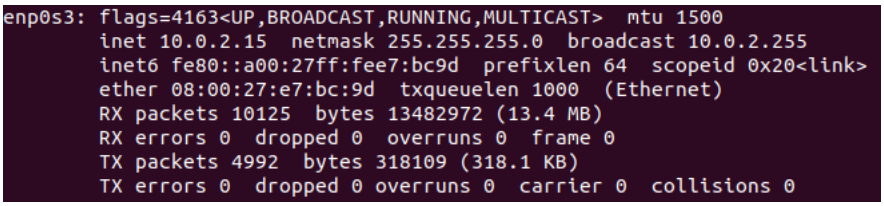

Figure 2: VM network card configuration

2.2 Testing the VM connectivity

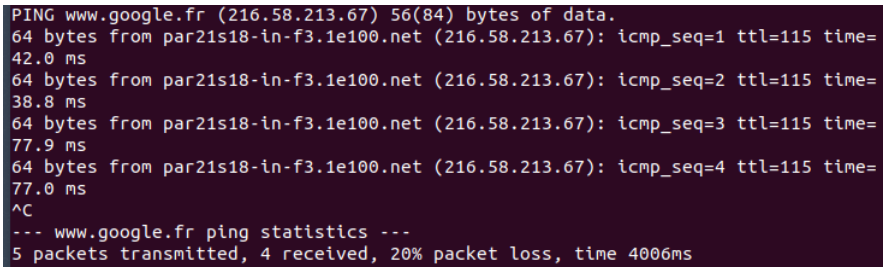

Figure 3: Ping from VM to Google

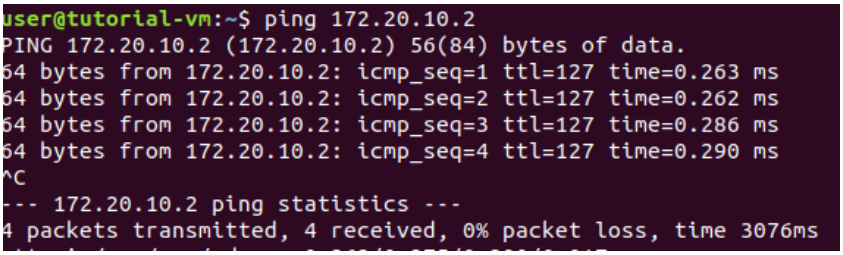

Figure 4: Ping from VM to host machine

We are able to ping the internet and the host machine. The VM has an IP in 10.0.2.0/24 in its private network. It is able to go on the internet by its gateway. We can see that the VM is also able to communicate with the host machine. To remember the VM is hosted on this machine, which means that its network card, even virtual, is the same as the host machine. The address in 172.20.10.2/28 is on the host network card, so even if it is a private address we are able to reach it from the VM because the gateway of the VM is on the same physical network card.

We are not able to ping from a neighbour host to the VM. This is because the IP address of the VM isn't routable. We can see it as an int simulating an IP address but this one isn't defined on the network : running two VM with this same image will result in two VMs with the same address. In fact, the hypervisor doesn't virtualize the network cards.

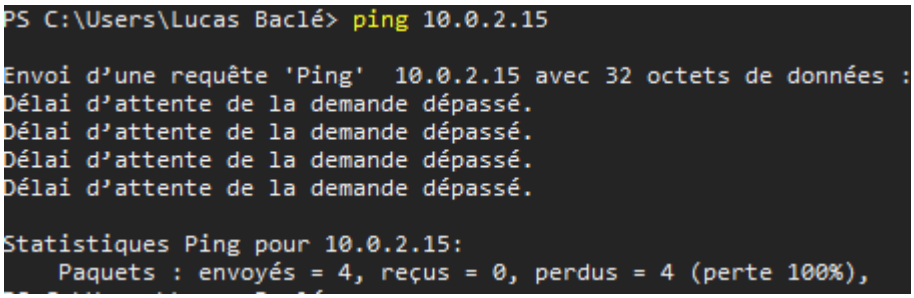

Figure 5: Ping from host machine to VM

We are not able to ping the VM from the host. The VM is in a private network on 10.0.2.0/24 and the host machine is in a private network on 172.20.10.0/28, so it is impossible to reach the private network of the VM from the private network of the host. Most importantly, the access to the VM on the host machine is on 192.168.56.0/24, which means that a ping with 10.0.2.15 as address will never go on this network, and so on never reach the VM.

2.3 Set up the "missing" connectivity

To fix the fact that we are unable to reach the VM from the host, we need to indicate that a packet with 192.168.56.1 as destination is in fact a packet with 10.0.2.15 as destination. With this technique we will be able to contact 10.0.2.15 reaching in fact 192.168.56.1. To do that we are going to use the port forwarding option in VirtualBox. This option allows us, for a given port (here 1234), to direct the packet through the right gateway. In other words when the network card will see 192.168.56.1:1234 as the destination address it will direct the packet on the virtual network card in 10.0.2.15:22.

Nom	Protocole	IP hôte	Port hôte	IP invité	Port invité
ssh	TCP	192.168.56.1	1234	10.0.2.15	22

Figure 6: Configuration of the port forwarding

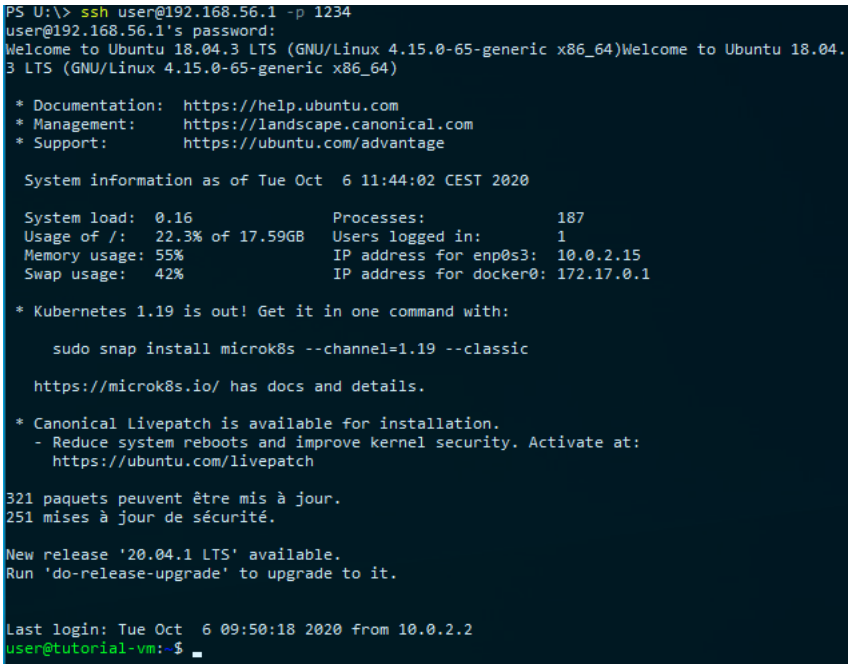

Figure 7: Test connection ssh host to VM

As a result we can see that we are now able to reach the VM and start an ssh session on it.

2.4 VM duplication

It is not possible to directly run the vm based on a disk copy because it uses the same identifier as the copied one. The hypervisor needs to have a unique identifier for each VM. In order to duplicate and deploy a VM, we need to use clonemedium.

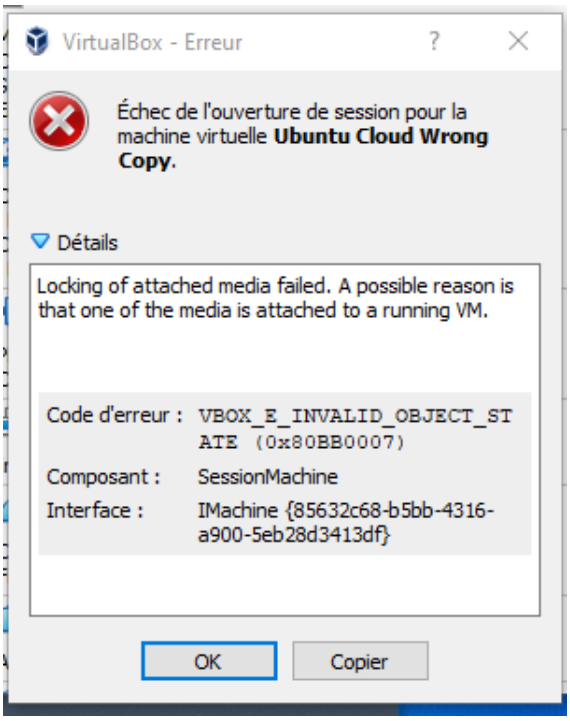

Figure 8: Error message from VM duplication

2.5 Docker containers provisioning

```
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 4787  bytes 15648933 (15.6 MB)
        RX errors 0  dropped 0  overruns 0  frame 0
        TX packets 4183  bytes 230868 (230.8 KB)
        TX errors 0  dropped 0 overruns 0  carrier 0  collisions 0
```

Figure 9: Container ifconfig

The docker container has 172.17.0.2 as IP address.

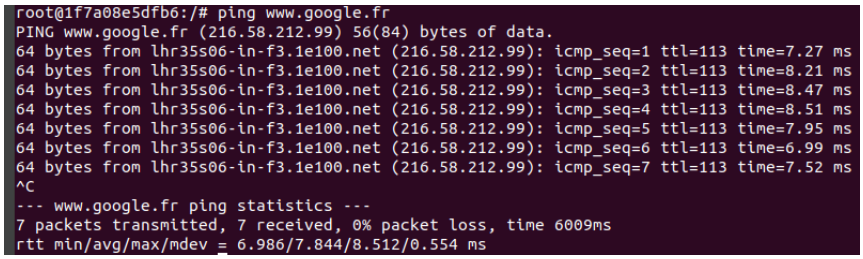

Figure 10: Ping from container to Google

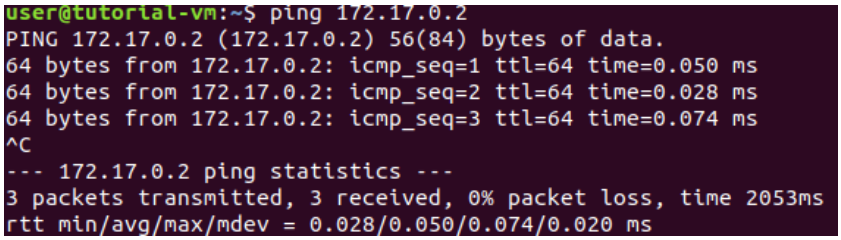

Figure 11: Ping from host to container

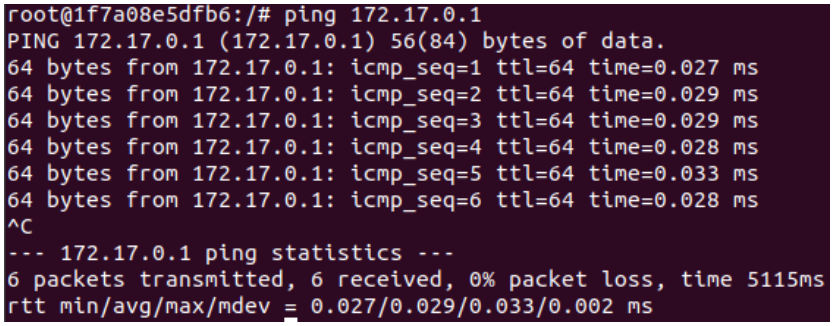

Figure 12: Ping from container to host

We can observe that the container is able to communicate with the internet and the host (VM in our case) while the host is also able to communicate with the container. Both the container, and the host machine which access its service, are on the same private network (172.17.0.0/16), this allows them to communicate without routing needed.

A snapshot takes a precise picture of a state of the container at a precise time. The image of the snapshot contains all the applications and libraries of the container at that precise time. This is why we still get nano when starting a Docker container with the snapshot.

3 OpenStack

3.1 CT creation and configuration on OpenStack

When we try to create our first VM, we can observe an error occurring. In fact, we do not have the rights required to deploy this VM on the INSA network (192.168.37.0/24). To avoid this, we first need to create a private network and deploy the VM into it later.

We create a private network (10.0.2.0/24) to host our VM. When we position our VM on this network an IP address is directly attributed to our VM (10.0.2.12). This means that a DHCP service is present on our network.

3.2 Connectivity test

To be able to reach internet from our private network, we place a router between our private network and INSA network. This router has an interface in our private network (10.0.2.254) and in INSA network.

Now we can reach internet from the VM but cannot reach the VM from the outside, this is because we are on a private network that cannot be routed. To solve this problem we are going to use a floating IP. Like the port forwarding in VirtualBox, INSA gives us a free IP address that we associate with our VM so we be able to reach it through this IP.

3.3 Snapshot, restore and resize a VM

We can resize our VM when it is down and when it is up, but this will cause a reboot in the second case. This show a great flexibility because we can easily adjust the resources that we need for each VM. However, the fact that the VM needs a reboot to do so can be a problem when we want to make changes on a VM which is in production.

We can realize snapshot of our VM in order to use it later to create new VM. This image keeps the hardware and software configuration of our VM. This can be useful in company when we need to replicate VM on different server or machine. We can realize different snapshot and install them on employee computer based on their need of software.

3.4 Deploy the Calculator application on OpenStack

In this part we create five VMs, one for each micro service and one for the frontend. Each VM is hosted on our private network. Only the frontend VM has a floating IP because this is the only VM that the user will be able to contact. The other VM does not need floating IP because they will contacted by the frontend which is on the same network.

In the configuration file of the frontend we indicate the different IP address given by the DHCP to each service. Now the frontend is able to use the services proposed by each VM. From a machine outside of the private network we realise

a REST request (arithmetic operations) on the frontend using its floating IP and we receive the answer with the result.

3.5 Client network oriented topology

Here we are going to simulate a client network topology. The idea is to have our micro services in our private network and the frontend in our client private network (cf figure 13). We can imagine that we develop an application for a client, we position the services in our private network to be able to update them easily, we deploy the application in our client network and the user needs to access this application through the client network.

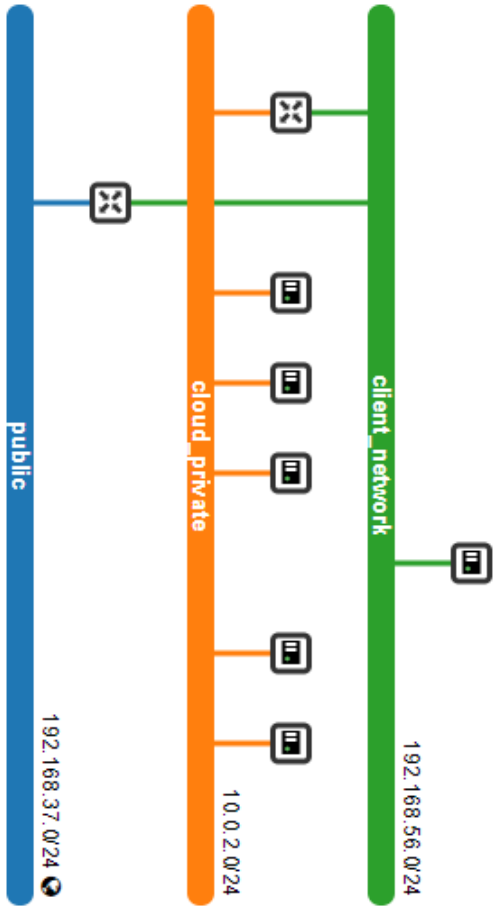

Figure 13: Network topology

We need two routers in this topology, one between the INSA network and

the client network and one between the client network and our network. The public gateway is configured like before, one interface in the INSA network which give access to the public network and one interface in the client network (192.168.56.254). The second router has also two interfaces, one in the client network (192.168.56.253) and one in our network (10.0.2.254).

In this configuration, the service does not work because our packet are not routed like we want. In the frontend VM routing table we only have a default gateway which is the public gateway. That means that all the packet try to go through the public gateway and we are not able access to our services. To solve this problem we need to add a route in the frontend routing table to indicate that the packet with 10.0.2.0/24 as destination need to pass by 192.168.56.253 router.

After associating a floating IP with our frontend VM, we are now able to contact the frontend from the outside and the frontend is able to use the services on our network. If we realise a REST request from an outside machine we receive the correct result of operation.