Cloud Computing: Adaptability and Autonomic Management

Lab 1 : Introduction to Cloud Hypervisors

Alexandre Cros - Louis Chauvet-Smart

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

The figure shows that the Virtual Machine and Container have different levels of virtualization: the container virtualizes on the OS level, and all apps in the container need to share the same operating system, while a virtual machine is virtualized on the hardware level, and has its own OS.

	Containers	**Virtual Machines**
Virtualisation cost, taking into consideration memory size and CPU	Lightweight and cheaper to host	Emulates an entire machine on the hardware level, so more expensive to virtualize.
Usage of CPU, memory and network for a given application	Pretty lightweight	Requires more CPU, memory and network. More control over the allocated resources for each VM though.
Security for the application (access right, resources sharing, etc.)	Process level isolation and shared OS, so some security flaws	Fully isolated machines, more secure.
Performances (response time)	Startup in milliseconds. short response time in relation to the CPU usage	Startup in minutes. For the same CPU usage from the host, higher response time since a whole machine has to be emulated
Tooling for the continuous integration support	Usually comes with development kits	No widespread development tools, at least by default
Flexibility, Dynamicity	Can easily change allocated resources for a given container or application, and requires minimal code to transfer and upload work	Hard to change the disk size and resources allocated to a virtual machine, and requires more work to transfer programs.

	Container	Virtual Machine
Developer point of view	The lightweight factor and flexibility of the container is very beneficial to developers, as it allows cheap and dynamic hosting for applications. The ability to share resources between different containers is also a great perk for app development, and the quick startup and response times are great to test and deploy programs. The dev kits provided with containers are also an obvious benefit to developers.	While heavier to host and less flexible, a virtual machine can be useful when the need comes to emulate entire machines, for example for low-level programming. If you need precise control over your network infrastructure, virtual machines also help, but in most cases a VM is overkill, and the high virtualization cost paired with the high boot and response times will make it a worse option for developers
System Administrator point of view	The security risks, low level of control over the hardware and network make containers a pretty bad tool for system administrators.	Virtual Machines have a lot of attributes system administrators are going to look for: every machine is fully isolated, they each have their own OS and therefore provide the associated tools, and the main downside of being heavyweight is usually less important to system admin who have to operate over very large systems and have more resources at their disposal.

2. Similarities and differences between the existing CT types

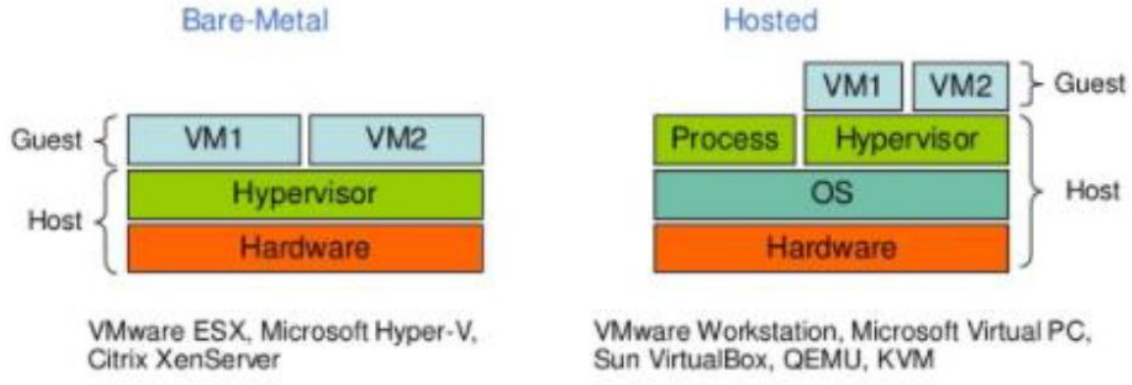
Let's detail our comparison criteria for the two CT we'll be studying :

- **Application isolation and resources, from a multi-tenancy point of view**
  Multitenancy describes the structure of a server hosting containers: a single instance of software (the host OS) serves multiple tenants (the containers). Application isolation and resources then describes how separated an application on a container is from other containers on the same server, and how its resources (shared or otherwise) are allocated
- **Containerization level (e.g. operating system, application)**
  Containerization means encapsulating software and all of its dependencies so that it can be run on any hardware. The containerization level therefore refers to the amount of components packed with the encapsulated software, and the level of abstraction the container will have in relationship to the host machine
- **Tooling (e.g. API, continuous integration, or service composition).**
  Tooling simply refers to the tools provided with the container service to aid the user.

These can take many forms, like development kits, migration tools, custom settings and more.

	Linux LXC	**Docker**
Application isolation and resources	Acts more like a light-VM, and is therefore more self-contained. It isn't constrained to a single application, and has a higher level of isolation than docker.	Works on the application level, and is less isolated. It shares resources between containers, and also uses more resources since it is more complex added to the LXC base.
Containerization level	LXC is virtualizing on OS level	Docker is based on LXC and adds higher level functionalities. Docker provide andengine to supervise the containerization, and virtualizes on the application level.
Tooling	As LXC is less popular, so the documentation isn't as thorough as docker. The LXC project provides base OS container templates and a comprehensive set of tools for container lifecycle management, but doesn't provide nearly as many development tools as docker.	Docker is running on application level and tools are centering on the Docker CLI. Docker Hub provides a public image access for frequently used application.

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



We can see from this diagram that type 1 hypervisors (left) and type 2 hypervisors (right) differ by the level on which they operate : type 1 is hardware level, while type 2 is OS level. Type 1 hypervisors have direct access to the hardware, and do not require an OS to be installed on the machine, they are therefore usually more efficient and better performing. They are also usually more secure, as they do not come with the security vulnerabilities of an OS.

Type 2 hypervisors are hosted on an underlying OS, and rely on that OS to perform certain tasks (managing network resources, allocate CPU usage, manage memory and storage). Type 2 hypervisors have the advantage of being supported by a variety of hardware. The main downside is the latency induced by having to interact with the OS.
Virtual Box is a type 2 hypervisor, and can be installed on almost any machine.
OpenStack is a cloud management system, so you get to choose which hypervisor to use. it can be either type 1 or type 2.

1. Tasks related to objectives 4 and 5

First part: Creating and configuring a VM

following the tutorial, we downloaded the virtual hard drive and created the VM on VirtualBox

Second part: Testing the VM connectivity

Using the ipconfig command, we can see both machines' IP address :
VM IP address : 10.0.2.15
Host IP adress : 10.32.46.110

Ping the outside with the VM :

```
user@tutorial-vm:~$ ping google.com
PING google.com (172.217.18.238) 56(84) bytes of data.
64 bytes from par10s10-in-f238.1e100.net (172.217.18.238): icmp_seq=1 ttl=115 time
=8.42 ms
64 bytes from par10s10-in-f238.1e100.net (172.217.18.238): icmp_seq=2 ttl=115 time
=9.19 ms
64 bytes from par10s10-in-f238.1e100.net (172.217.18.238): icmp_seq=3 ttl=115 time
=9.54 ms
```

Ping from another machine to the hosted VM :

```
C:\Users\louis>ping  10.0.2.15

Pinging 10.0.2.15 with 32 bytes of data:
Request timed out.
Request timed out.
Request timed out.
Request timed out.

Ping statistics for 10.0.2.15:
    Packets: Sent = 4, Received = 0, Lost = 4 (100% loss),
```

Ping from host to the VM :

```
C:\Users\Alex>ping 10.0.2.15

Pinging 10.0.2.15 with 32 bytes of data:
Request timed out.
Request timed out.

Ping statistics for 10.0.2.15:
    Packets: Sent = 2, Received = 0, Lost = 2 (100% loss),
```

this is exactly what we expected from the NAT connexion, as the machine connects to a private network managed by the hypervisor

Third part: Set up the “missing” connectivity

To allow the virtual machine to be accessed from an external network despite the NAT protocol, we set up a port forwarding rule: we redirect requests received by the host on a specific port (we used 1234 but we can use any port over 1024) to a specific port on the virtual machine: we redirect to port 22, the ssh port. Therefore, by sending a request on our host’s 1234 port, we send an ssh request to the virtual machine and connect to it from an external request.

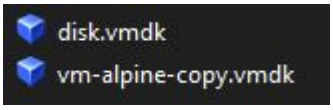
Name	Protocol	Host IP	Host Port	Guest IP	Guest Port
SSH	TCP	10.32.46.110	1234	10.0.2.15	22

```
C:\Users\Alex>ssh -p 1234 user@10.32.46.110
The authenticity of host '[10.32.46.110]:1234 ([10.32.46.110]:1234)' can't be established.
ECDSA key fingerprint is SHA256:RwbyEulL7lk++LikmWRQzNhQLfvIBjMtiMQV/WHg8lo.
Are you sure you want to continue connecting (yes/no)? yes
```

Fourth part: VM duplication

```
C:\Program Files\Oracle\VirtualBox>VBoxManage clonemedium "C:\Users\Alex\OneDrive - insa-toulouse.fr\Cours\
5ISS\Cloud\Environment\disk.vmdk" "C:\Users\Alex\OneDrive - insa-toulouse.fr\Cours\5ISS\Cloud\Environment\v
m-alpine-copy.vmdk"
VBoxManage.exe: error: Failed to lock source media 'C:\Users\Alex\OneDrive - insa-toulouse.fr\Cours\5ISS\Cl
oud\Environment\disk.vmdk'
VBoxManage.exe: error: Details: code VBOX_E_INVALID_OBJECT_STATE (0x80bb0007), component MediumWrap, interf
ace IMedium, callee IUnknown
VBoxManage.exe: error: Context: "CloneTo(pDstMedium, ComSafeArrayAsInParam(l_variants), NULL, pProgress.asO
utParam())" at line 1068 of file VBoxManageDisk.cpp

C:\Program Files\Oracle\VirtualBox>VBoxManage clonemedium "C:\Users\Alex\OneDrive - insa-toulouse.fr\Cours\
5ISS\Cloud\Environment\disk.vmdk" "C:\Users\Alex\OneDrive - insa-toulouse.fr\Cours\5ISS\Cloud\Environment\v
m-alpine-copy.vmdk"
0%...10%...20%...30%...40%...50%...60%...70%...80%...90%...100%
Clone medium created in format 'VMDK'. UUID: 9f774591-9df2-496a-8ba8-d06fb20665ab
```



Command has been executed and the disk was copied succesfully.

Fifth part: Docker containers provisioning

got an image of ubuntu:

```
user@tutorial-vm:~$ sudo docker pull ubuntu
[sudo] Mot de passe de user :
Using default tag: latest
latest: Pulling from library/ubuntu
d72e567cc804: Pull complete
0f3630e5ff08: Pull complete
b6a83d81d1f4: Pull complete
Digest: sha256:bc2f7250f69267c9c6b66d7b6a81a54d3878bb85f1ebb5f951c896d13e6ba537
Status: Downloaded newer image for ubuntu:latest
```

executed an instance of the ubuntu image

```
user@tutorial-vm:~$ sudo docker run --name ct1 -it ubuntu
```

installed connectivity tools

```
root@5d0c99827867:/# apt-get -y update && apt-get -y install net-tools iputils-ping
```

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

    1. What is the Docker IP address.

        ```
        inet 172.17.0.2
        ```

    2. Ping an Internet resource from Docker

```
root@5d0c99827867:/# ping riot.de
PING riot.de (88.198.0.110) 56(84) bytes of data.
64 bytes from mx03.droids.de (88.198.0.110): icmp_seq=1 ttl=50 time=29.8 ms
```

it works!

    3. Ping the VM from Docker

```
root@5d0c99827867:/# ping 10.0.2.15
PING 10.0.2.15 (10.0.2.15) 56(84) bytes of data.
64 bytes from 10.0.2.15: icmp_seq=1 ttl=64 time=0.046 ms
```

it also works!

    4. Ping the Docker from the VMS

```
user@tutorial-vm:~$ ping 172.17.0.2
PING 172.17.0.2 (172.17.0.2) 56(84) bytes of data.
64 bytes from 172.17.0.2: icmp_seq=1 ttl=64 time=0.028 ms
```

it also also works!

These results are expected : unlike the VM who virtualizes on a hardware level, docker (a container) virtualizes on the OS level. The container can therefore access the resources and the network of its host, as the encapsulation is lighter.

Executing new instance of the ubuntu docker and installing nano:

```
user@tutorial-vm:~$ sudo docker run --name ct2 -p 223:22 -it ubuntu
[sudo] Mot de passe de user :
root@aa0f58a16f68:/# apt-get -y update && apt install nano
```

checking the ID of the docker instances

```
user@tutorial-vm:~$ sudo docker ps
[sudo] Mot de passe de user :
CONTAINER ID        IMAGE               COMMAND             CREATED
STATUS              PORTS               NAMES
aa0f58a16f68        ubuntu              "/bin/bash"         2 minutes ago
Up About a minute   0.0.0.0:223->22/tcp   ct2
5d0c99827867        ubuntu              "/bin/bash"         21 minutes ago
Up 21 minutes                             ct1
```

made a snapshot of CT2

```
user@tutorial-vm:~$ sudo docker commit aa0f58a16f68 container2:version1
sha256:89818b70b3c392ed6766b9c3db1f116322d935366e5daff64d180990a15ae4b9
```

stopped and terminated CT2

```
user@tutorial-vm:~$ sudo docker stop aa0f58a16f68
aa0f58a16f68
user@tutorial-vm:~$ sudo docker rm aa0f58a16f68
aa0f58a16f68
```

CT2 is terminated

```
user@tutorial-vm:~$ sudo docker ps
CONTAINER ID        IMAGE               COMMAND             CREATED
STATUS              PORTS               NAMES
5d0c99827867        ubuntu              "/bin/bash"         31 minutes ago
Up 31 minutes                           ct1
```

Checking available docker images:

```
user@tutorial-vm:~$ sudo docker images
REPOSITORY          TAG                 IMAGE ID            CREATED             SIZE
container2          version1            89818b70b3c3        3 minutes ago       97.9MB
ubuntu              latest              9140108b62dc        11 days ago         72.9MB
```

We can see the snapshot we made of CT2 is still available

We then create a third container from the ct2 snapshot we made:

```
user@tutorial-vm:~$ sudo docker run --name ct3 -it container2:version1
```

Nano is already installed on the container

```
user@tutorial-vm:~$ sudo docker run --name ct3 -it container2:version1
root@cd89bc1d77fc:/# nano --help
Usage: nano [OPTIONS] [[+LINE[,COLUMN]] FILE]...

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

 Option         Long option             Meaning
 -A             --smarthome             Enable smart home key
```

This is explained by the fact that containers use shared resources on the host machine. When we installed nano in container2, we downloaded the files in the shared resources and said that container2 had the right to access them. When we created the snapshot, we copied those rights, and our new container was to access the files and had nano properly configured.

to create a recipe, we first created our dockerfile:

```
user@tutorial-vm:~$ touch myDocker.dockerfile
```

```
FROM ubuntu
RUN apt update -y
RUN apt install -y nano
CM ["/bin/bach"]
~
```

(CM was a typo, corrected to CMD)

Finally, we build the image in the VM

```
user@tutorial-vm:~$ sudo docker build -t container3:version2 -f myDocker.dockerfile .
Sending build context to Docker daemon  757.2kB
Step 1/4 : FROM ubuntu
```

```
Successfully built 009549484df8
Successfully tagged container3:version2
```

OPENSTACK

First part: CT creation and configuration on OpenStack

it looks like the instance is not launching !

> **Error:** Failed to perform requested operation on instance "tp", the instance has an error status: Please try again later [Error: Exceeded maximum number of retries. Exceeded max scheduling attempts 3 for instance 456691b6-4079-40af-896f-e2a4f9afaf83. Last exception: Binding failed for port c71c534d-1a3b-407b-8bde-73a701dfd2f6, please check neutron logs for more information.].

The VM can't launch because the network can't give it an IP address: because of the public network's security settings, it doesn't accept the new VM as part of the network and doesn't give it an IP address. We need to setup our private network to be able to add our VMs to it.

To do so, we add two rules to the security group: SSH and ICMP. We then setup the network topology, with a private network containing our VM, and a gateway between the public and private networks

.

Second part: Connectivity test

Before setting up the private network, we still aren't able to launch an instance of the VM: we can't change the security settings of the public network (obviously), so we create our own private network that we can customize to accept these new rules. To be able to communicate with the public network though, we setup a gateway to allow communication between the two networks.

Once the network is set up, we can see an IP address has been attributed to the VM. We can ping the public network from the VM (both google and insa machines), but we can't yet ping the VM from the public network.

Third part: Snapshot, restore and resize a VM

- Resize a running VM from the Instances tab. What do you observe? Write down your comments.

The size of the VM changes, no special remarks. The IP address doesn't change.

- Shutdown the VM and redo the same operation. Write down your comments.

The task changes to "preparing to resize or migrate", and then the resize fails. This is because we are downsizing the flavour, so we don't have enough resources to start the VM up again. The resize then gets aborted. However if we try to resize it to a larger size, it goes through without any issues. This is true whether the VM is active or not.

- The 2 previous operations highlight the flexibility and the agily that could be provided thanks to the virtualization setting as it was previously explained during the lectures. However, this does highlight an existing technical limitation (related to OpenStack). What is it? Can you propose a hint/way to address it?

These tests reveal that OpenStack doesn't really check whether the resizing of the instance allows the VM to properly boot. Even worse, downsizing an instance could lead to damage to the VM's files. This problem could be solved by checking the system requirements of the VM in each instance before resizing, and restricting downsizing if it would cause damage to the VM.

- Make a snapshot of the VM. Elaborate on the differences between the included material/software within the original image and the newly created snapshot
- Restore the VM from the last backup. Write down your comments.

It seems like all the machine's parameters and software are identical, however the network settings aren't exactly the same: if we create a VM with the snapshot, the IP address isn't the same as the original machine. This shows that they are considered as two separate machines by the network.

OpenStack client installation

```
user@tutorial-vm:~/Téléchargements$ ls
5ISS-Cloud-B21-openrc.sh
user@tutorial-vm:~/Téléchargements$ source 5ISS-Cloud-B21-openrc.sh
Please enter your OpenStack Password for project 5ISS-Cloud-B21 as user lchauvet:
user@tutorial-vm:~/Téléchargements$ openstack
```

```
(openstack) project list --help
usage: project list [-h] [-f {csv,json,table,value,yaml}] [-c COLUMN]
                    [--quote {all,minimal,none,nonnumeric}] [--noindent]
                    [--max-width <integer>] [--fit-width] [--print-empty]
                    [--sort-column SORT_COLUMN] [--domain <domain>]
                    [--user <user>] [--my-projects] [--long]
                    [--sort <key>[:<direction>]]
```

Part three: Deploy the Calculator application on OpenStack

We had to install ‘sync-request’ by using npm with the following command :

npm install syn-request

We also had to change the listening port in the source code of *CalculatorService.js* from 80 to 50000.

```
const PORT = process.env.PORT || 50000;
```

As the local IP address of services are different, we need to change those IPs in CalculatorService.js as following :

```
const SUM_SERVICE_IP_PORT = 'http://192.168.0.79:50001';
const SUB_SERVICE_IP_PORT = 'http://192.168.0.22:50002';
const MUL_SERVICE_IP_PORT = 'http://192.168.0.64:50003';
const DIV_SERVICE_IP_PORT = 'http://192.168.0.186:50004';
```

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

	Instance Name	Image Name	IP Address
☐	calculatorService	calcSnap	192.168.0.90, 192.168.37.155
☐	divService	calcSnap	192.168.0.186, 192.168.37.51
☐	MulService	calcSnap	192.168.0.64, 192.168.37.177
☐	subService	calcSnap	192.168.0.22, 192.168.37.183
☐	sumService	ubuntu4CLV	192.168.0.79, 192.168.37.13

Port list for each application :

sumService: 50001

subService: 50002

mulService: 50003

divService: 50004

calculatorService: 50000

Because of the port listening we needed to add a new rule to “Security Group” in order to allow packet transfert on port range 50000 to 50004 as following :

☐	Ingress	IPv4	TCP	50000 - 50004	0.0.0.0/0

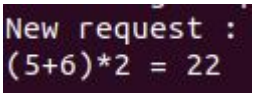
We can now request all the services with the following command :

```
CalculatorService : $ curl -d "(5+6)*2" -X POST http://<ip>:50000
SumService :        $ curl -d "2 3" -X POST http://<ip>:50001
SubService :        $ curl -d "5 3" -X POST http://<ip>:50002
MulService :        $ curl -d "12 3" -X POST http://<ip>:50003
DivService :        $ curl -d "15 5" -X POST http://<ip>:50004
```

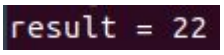
Request example :

```
curl -d "(5+6)*2" -X POST http://192.168.37.155:50000
```

The service receives the request and does the following job :



Finally, the client received the answer :

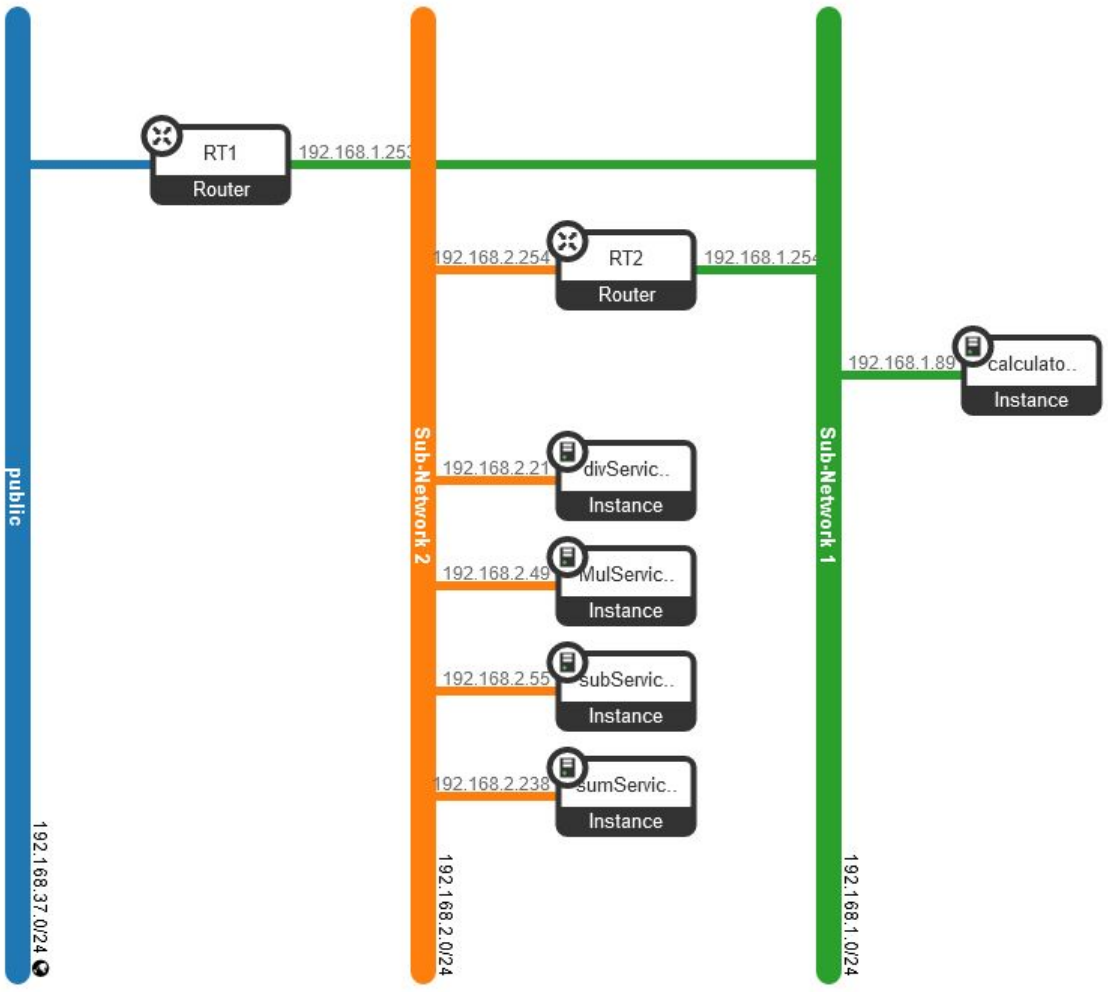


When we change the source code while the service is running (adding a console.log) , it receives the request, does the operation, but doesn't send the result back. If we comment that extra line

Part four: Automating the deployment using docker containers, exporting them into a json file and using the node docker api to parse that file and deploy the services would've been fun but very time consuming, so we've opted to skip this part to be able to do the lab's last part. We instead talked with other groups about their version of this program.

Part 10

We created the network on openstack following the topology given in the tutorial



We used the virtual machines with the services already running on them to avoid having to configure them an extra time

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

The front end machine and the calculating machines can't communicate. Since they're on different networks, we need to set up routes so that the different machines know how to reach each other despite being in different networks.

To do so, we must first add a route that tells our front end machine to access network2 any time it tries to communicate with one of the 4 services.

The default route in network1 also seemed to be a problem, as after setting everything up, we could ping the router but not the calculator. By rewriting the default route manually, we

fixed this issue and were able to send requests to the service.

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

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

We also need to specify the services' new IP addresses in the calculator's source code

```
const SUM_SERVICE_IP_PORT = 'http://192.168.2.238:50001';
const SUB_SERVICE_IP_PORT = 'http://192.168.2.55:50002';
const MUL_SERVICE_IP_PORT = 'http://192.168.2.49:50003';
const DIV_SERVICE_IP_PORT = 'http://192.168.2.21:50004';
```

When the calculator service is requested, it replies with the right result :

```
curl -d "(5+6)*2" -X POST http://192.168.37.82:50000
result = 22
```

The calculator service receive the request and answer as following :

```
New request :
(5+6)*2 = 22
```