Lab 1 : Introduction to Cloud Hypervisors

S. Yangui (INSA/LAAS)

Access link : tiny.cc/TP_CloudComputing

Introduction

The general purpose of this lab is to enhance your knowledge of concepts and technologies for virtualization techniques. These techniques are used in IT environments that support the provisioning (i.e. development, deployment, management) of novel software systems (with their potential constraints such as QoS and security). These environments are typically dynamic and distributed.

The tutorials will be evaluated based on the work performed during lab hours, but also on your personal and dissertation efforts. The students must write an online synthesis in parallel during the lab hours. Involvement during lab hours will also be considered in the final mark.

The provided documentation resources are listed below:

- OpenStack (Rocky distribution) user guide
  - https://docs.openstack.org/rocky/
- VirtualBox virtualization software manual
  - http://download.virtualbox.org/virtualbox/UserManual.pdf
- *Virtual appliance library*
  - *http://www.turnkeylinux.org/*

Lab objectives

The purpose of this lab is to discover and test the basic functionalities (features) that cloud service providers could offer to end-users at the infrastructure level (IaaS perspective).
The lab objectives are detailed in what follows:

- Objective 1: Recognize the fundamental differences between the types of hypervisors' architectures (type 1/type 2).
- Objective 2: Recognize the fundamental differences between the two main types of

virtualisation hosts[1], i.e. *virtual machines* (VM) and *containers* (CT).

- Objective 3: Evaluate the two different modes of network connection for VM and CT (i.e. NAT and Bridge modes).
- Objective 4: Operate VirtualBox VMs provisioning, in NAT mode, and make appropriate configurations so that VMs can access and be accessed through the Internet
- Objective 5: Operate Docker containers provisioning, and make appropriate configurations so that containers can access and be accessed through the Internet
- Objective 6: Operate proper VMs provisioning with OpenStack and manage the networking connectivity
- Objective 7: Test and evaluate the supported management operations on the virtualization hosts (e.g. snapshot, restore, backup, resize).
- Objective 8: Use the OpenStack API in order to automate the operations described in objectives 4, 5, 6 and 7.
- Objective 9: Implement a simple orchestration tool in order to provision a Web 2-tier application.
- Objective 10 : Make up a specific network topology on OpenStack
- Objective 11: Configure and deploy this network topology

Lab format

The lab is organized in two main parts:

1. The first part is theoretical. You first need to carefully read this part and assimilate to recall all the concepts introduced during the lecture. Then, when required, you need to write down your answers in the shared online document.

2. The second part is practical (hands-on).

[1] According to the ETSI terminology.

Theoretical part
(objectives 1 to 3)

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

There are two types of virtualisation hosts (i.e. VMs and CTs). These hosts are depicted in Figure 1.

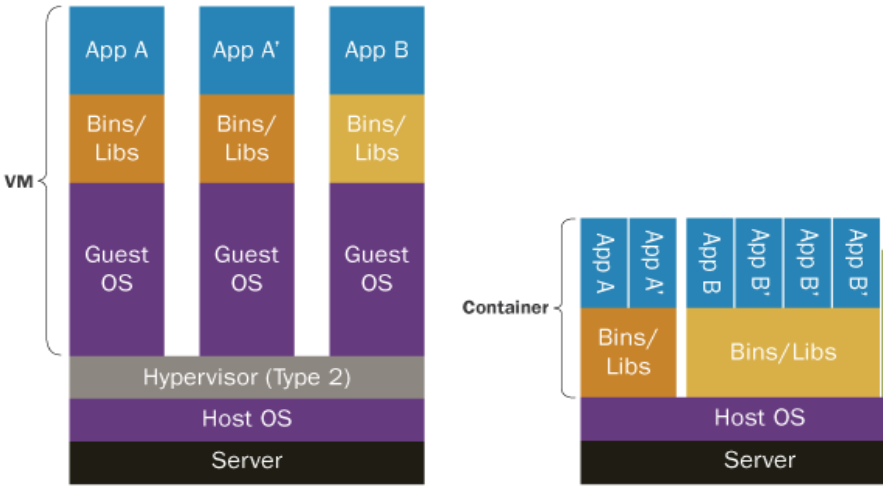


Figure 1: VM vs CT

Tasks (You need to write down your answers in the shared document):

- Understand the figure above, and elaborate on it.
- 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.
  For each one of these perspectives, the comparison should be based on the following criteria:
    - virtualization cost, taking into consideration memory size and CPU,
    - Usage of CPU, memory and network for a given application,
    - Security for the application (access right, resources sharing, etc.),
    - Performances (response time),
    - Tooling for the continuous integration support

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 the one displayed in Figure 2.

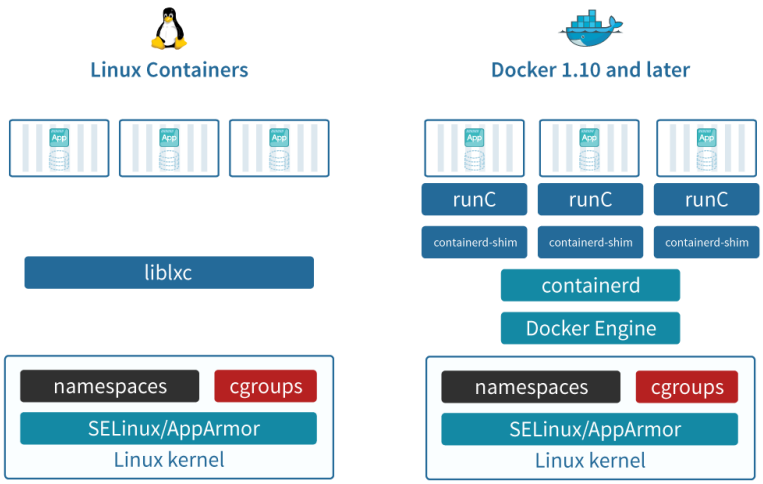


Figure 2 : Linux Lxc vs Docker

These technologies can however be compared based on the following criteria (non-exhaustive list):

- Application isolation and resources, from a multi-tenancy point of view,
- Containerization level (e.g. operating system, application),
- Tooling (e.g. API, continuous integration, or service composition).

Tasks (You need to write down your answers in the shared document):

- Elaborate on the proposed criteria,
- By doing your own research on the Web, classify the existing CT technologies (LXC, Docker, Rocket, …) based on these criteria, and eventually, on additional criteria that you need to identify by yourself.

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

There are two main categories of hypervisors, referred to as type 1 and type 2.

Tasks (You need to write down your answers in the shared document):

- Based on the lecture's slides (slides 14, 15 and 16), summarize and elaborate on each of these types.
- Identify which architecture type VirtualBox and OpenStack belong to.

4. Difference between the two main network connection modes for virtualization hosts

Tasks (No writing efforts required)

With some hypervisors, multiple possibilities exist to connect a VM/CT to the Internet, via the host machine (in this case your desktop). The two main modes are the following:

- *NAT mode*: It is the default mode. It does not require any particular configuration. In this

mode, the VM/CT is connected to a private IP network (i.e. a private address), and uses a virtual router (managed by the hypervisor) running on the host machine to communicate outside of the network:
    - This router executes what is equivalent to a NAT function (only *postrouting*) allowing the VM to reach the host machine or the outside;
    - However, the VM cannot be reached from the host (and by any other VM hosted on the same machine): to make it accessible from outside, it is necessary to deploy port forwarding (*prerouting*).
- *Bridge mode*, the most widely used (yet not systematically), in which the VM/CT sees itself virtually connected to the local network of its host (see Figure 3): it has an IP address identifying it on the host network, and can access the Internet (or is accessed) as the host.

NB: Other less used modes exist, such as Private Network in VirtualBox, that you can try out.

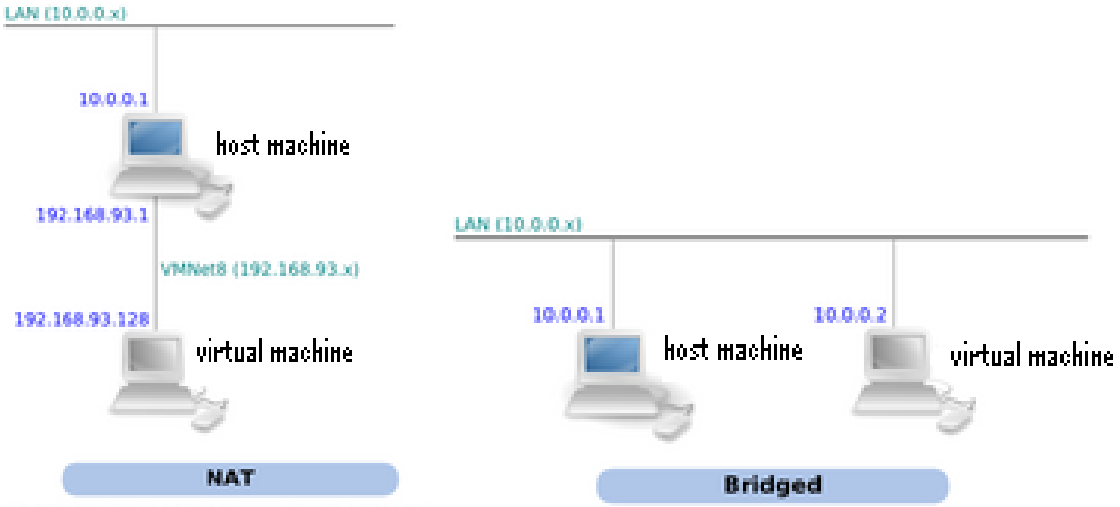


Figure 3 : Mode NAT vs Mode Bridge (the most usual)

Practical part
(objectives 4 to 7)

For each item thereafter, you must make a short written report on the shared document (with snapshots when applicable).

1. Tasks related to objectives 4 and 5

Creating a VirtualBox VM (in NAT mode), and setting up the network to enable two-way communication with the outside

In this part, you will use the VirtualBox hypervisor (Type 2) in NAT mode to connect a VM to the network. The bridge mode will be used in the second part of the lab, with another hypervisor (OpenStack) - Type 1.

First part: Creating and configuring a VM

- Copy the virtual hard drive of the Ubuntu operating system (**Link**)
- Unzip the archives.
- Start VirtualBox (installed on Windows session).
- In VirtualBox, create a VM - **Type Linux / Ubuntu 64 bits** and configure it:
  - You need to associate the following resources to it: RAM (for instance 1024 Mo), hard drive (provided) and CPU (1 core max),
  - Associate peripherals (e.g. network board, CD readers, USB ports), choosing a NAT mode network board (the private IP address is automatically attributed).
- Run the VM. The authentication information are the following:
  - username : **osboxes**
  - password: **osboxes.org**

Second part: Testing the VM connectivity

Identify the IP address attributed to the VM using ifconfig (in Linux command line), and compare it to the host address (ip config in the command line). What do you observe?

L'inverse

With a ping command (or any other command of your choice) :

- Check the connectivity from the VM to the outside. What do you observe?
- Check the connectivity from your neighbor's host to your hosted VM. What do you observe?
- Check the connectivity from your host to the hosted VM. What do you observe?

Third part: Set up the "missing" connectivity

In this part, we propose to fix the issues identified in the previous part, for a dedicated application

(e.g. SSH, port 22). To that end, we are going to use the *Port Forwarding* technique. You need to add a new forwarding rule to the management interface of the network board provided by VirtualBox,

For the testing, you need to:

- *install open ssh server in the VM*

```
$ sudo apt-get install openssh-server
```

- *Use PuTTy as a SSH client or the following command*

```
$ ssh login@host -p port
```

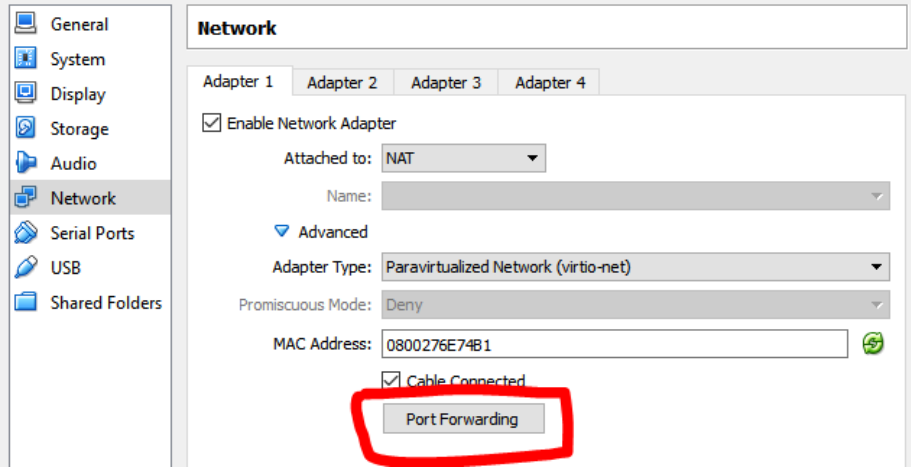


Figure 4 : Port forwarding

Fourth part: VM duplication

To create a new (clone) VM with the same disk file:

- Run the following command:

```
VBoxManage clonemedium "chemin\vers\disk.vdi" "chemin\vers\disk-copy.vdi"
```

  - *NB: VBoxManage is not globally declared. It must be invoked with C:\Program Files\Oracle\VirtualBox\VBoxManage.exe in Windows.*

- Create the new VM with the disk-copy.vmdk file

Fifth part: Docker containers provisioning

For practical reasons, the target Docker environment will be deployed over VirtualBox VM since you will have all the admin privileges on these VMs. Indeed, provisioning containers over VMs is technically possible but remains very rare in practice. Containers are usually deployed in bare-metal on physical hosts like it is the case with VMs.

We will first install the Docker Engine on the VM then we will use it to create and handle Docker containers.

Docker Engine Setup

First, update your existing list of packages

```
$ sudo apt update
```

Next, install a few prerequisite packages which let *apt* use packages over HTTPS

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

Then add the GPG key for the official Docker repository to your system

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

Add the Docker repository to APT sources

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

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

```
$ sudo apt update
```

Make sure you are about to install from the Docker repo instead of the default Ubuntu repo

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

Notice that docker-ce is not installed, but the candidate for installation is from the Docker repository for Ubuntu 22.04 (jammy).

Finally, install Docker

```
$ sudo apt install docker-ce
```

Docker should now be installed, the daemon started, and the process enabled to start on boot. Check that it's running.

```
$ sudo systemctl status docker
```

Installing Docker now gives you not just the Docker service (daemon) but also the `docker` command line utility, or the Docker client. We'll explore how to use the `docker` command in the following section.

Docker Containers Provisioning

Now, we are ready to provision Docker nodes. Using the `docker` command consists of passing it a chain of options and commands followed by arguments. The syntax takes this form

```
$ docker [option] [command] [arguments]
```

To view all available subcommands, type

```
docker docker-subcommand --help
```

To view system-wide information about Docker, use

```
docker info
```

As for Docker nodes provisioning, you need to follow the steps below:

Get an ubuntu image:

```
$ sudo docker pull ubuntu
```

Execute an instance of the ubuntu image (CT1):

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

Install the required connectivity testing tools :

```
$ sudo apt-get -y update && apt-get -y install net-tools iputils-ping
```

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

1. What is the Docker IP address?
2. Ping an Internet resource from Docker
3. Ping the VM from Docker
4. Ping the Docker from the VM
5. Elaborate on the obtained results

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

```
$ sudo docker run --name ct2 -p 2223:22 -it ubuntu
```

Install nano (text editor) on CT2.

```
[CT2] $ apt-get -y update && apt install nano
```

Make a snapshot of CT2 :

```
$ sudo docker commit %ID_de_CT2  %REPO:%TAG
```

Use `$ sudo docker ps`. to obtain the CT2 ID.

The "commit" command documentation is available via this link.

Stop and terminate CT2

```
$ sudo docker rm %ID_de_CT2
```

List the available Docker images in the VM

```
$ sudo docker images
```

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

```
$ sudo docker run --name ct3 -it %REPO:%TAG
```

Do you still have nano installed on CT3? Explain the reason in the shared document.

Docker enables "recipes" sharing to create persistent images (as a second alternative of snapshots). To make a proper recipe, you need to write a *Dockerfile (myDocker.dockerfile)*:

```
FROM ubuntu

RUN apt update -y

RUN apt install -y nano

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

Then, you need to build the image in the VM

```
$ sudo docker build -t %REPO:%TAG -f myDocker.dockerfile .
```

2. Expected work for objectives 6 and 7

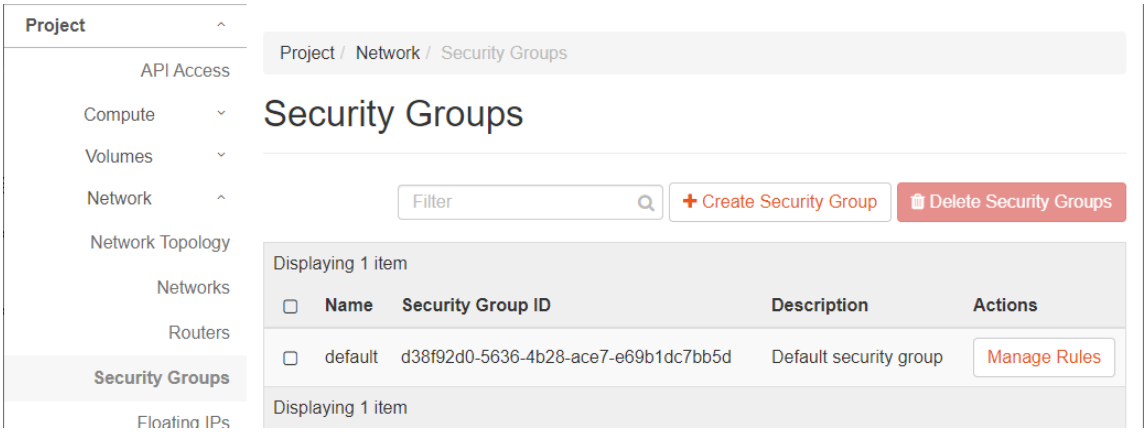
First part : CT creation and configuration on OpenStack

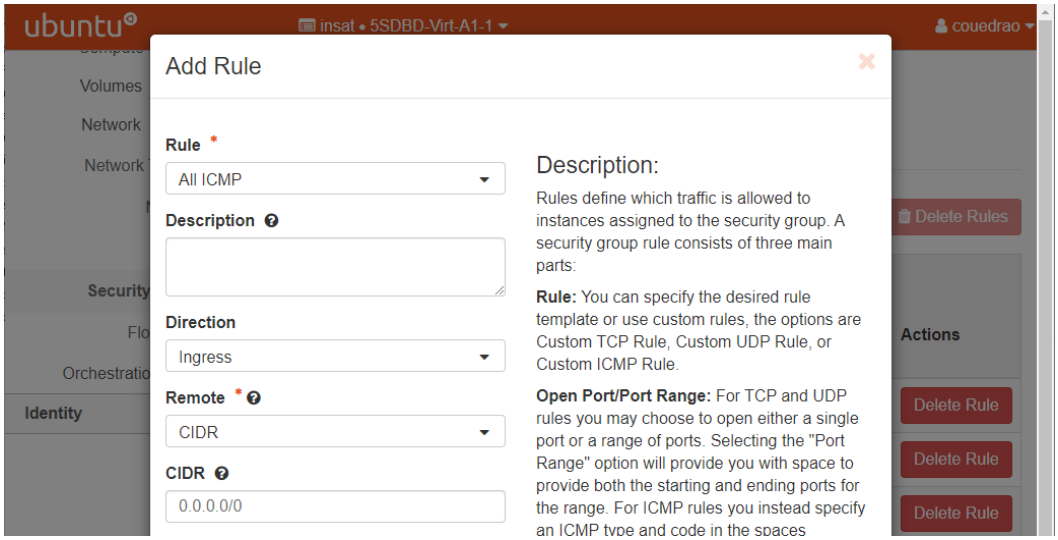
- Connect to OpenStack Web interface
    - *NB : If you are connecting from outside INSA, the use of INSA VPN is required. You download and get the VPN configuration from this link.*
- Authenticate with your INSA login/pwd using "**INSA**" group/domain
- Make sure you are connected to your default project on OpenStack.
- Select the Instances tab and create a VM (Launch Instance) using the available "ubuntu4CLV" or the alpine images and the "small2" flavor. Please note that you could also use the image of the VM that you did export from VirtualBox. The VM will automatically start and all related information is displayed in the dashboard. What do you observe?
- Connect to the VM using the console feature

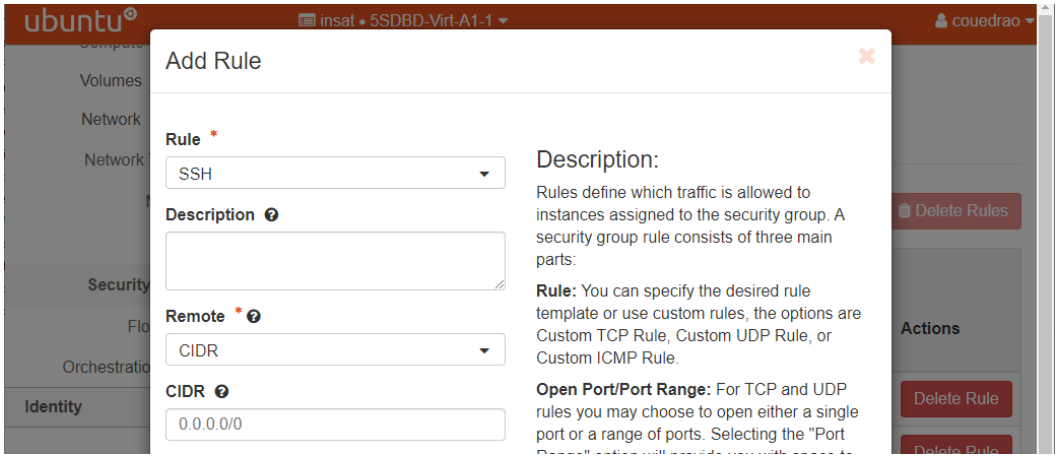
Configuring the security rules in OpenStack

By default, OpenStack blocks the network traffic on the virtualized private network. This traffic could be managed using specific security rules that are specific to each project. You need at least to authorize the ping (ICMP) and SSH traffic for the rest of the lab.

- Go to the "Security Group" section in the "Network" tab
- Manage the rules of your default security group in order to add/authorize the required ICMP (Ingress) and SSH.







Second part: Connectivity test

The VM IP address that was given by the hypervisor is displayed on the dashboard (Instances tab). What do you think about this address? Write down your comments on the shared document.

- Using a Ping (or any other appropriate command) :
  - Check the connectivity from the VM to the desktop. Write down your comments
  - Check the connectivity from the desktop to the VM. Write down your comments.

- You can refer to the graphical schema that depicts the network and all instantiated resources in the Network topology section of the Networks tab in order to help you to understand the networking and routing mechanisms. Write down your comments and identify the problem.
- In order to solve this problem, we propose to create a private network where all created

VMs will be connected to. This network will be connected to the public network (and thus to Internet) through a router that will implement a gateway role between the 2 networks. The target topology is depicted in Figure 5.

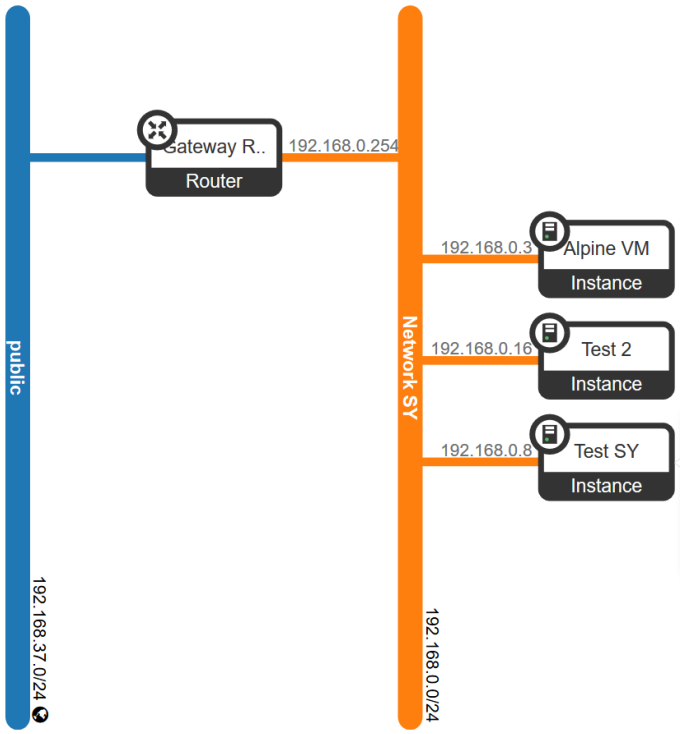


Figure 5 : Target Network Topology

- Create a new network from the Networks section in the Network tab. Specify an IP range for your network, as well as, the IP address for the gateway.
- Double-check with the associated Network topology and modify/adapt if necessary.
- Re-create a new VM and select the newly created private network during the creation process.
- Double-check with the associated Network topology and modify/adapt if necessary
- Configure the VM in order to connect it to the Internet. To that end, from the Routers section in the Network tab, you need to create and configure the router that will implement the gateway between the 2 networks as it is depicted in Figure 5.
- Configure the VM in order to make it accessible from outside. To that end, you need to create and associate a floating IP to it. Floating IP can be created from the Floating IPs section in the Network tab.
- Test the connectivity of the VM from outside using an SSH client. Write down your comments.

  Note that, in order to facilitate future operations on VMs, you can create and generate SSH key and associate them to the VMs during the creation process.

This can be done from the Key pairs section in the Compute tab.

Third part: Snapshot, restore and resize a VM

- Resize a running VM from the Instances tab. What do you observe? Write down your comments.
- Shutdown the VM and redo the same operation. Write down your comments.
- 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?
- 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.

3. Expected work for objectives 8 and 9

Part one : OpenStack client installation

OpenStack comes with a command line client that enables calling the remote REST operations.

- Install the client from **your Ubuntu VM on VirtualBox**

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

- Configure the client with the appropriate variables (OpenStack address, port, used project, etc.). To that end, you need to generate and download (from the dashboard) the RC v3 file that contains all the required information and then, locally execute it from a terminal

```
$source fichier_rc.sh
```

- Start the client

```
$ openstack
```

- Display the help

```
(openstack) help
```

- Display the help of a specific command

```
(openstack) project list --help
```

Part two : Web 2-tier application topology and specification

In this part, we propose to deploy a 2-tier Web application on OpenStack. The application implements a calculator that handles arithmetic operations (i.e. addition, subtraction, multiplication and division) of integer numbers. Each one of these operations is implemented with a NodeJs microservice. In addition, there is a fifth micro-service that implements the application endpoint. Specifically, this service will be listening and receiving the prospective requests when starting the application. Requests are HTTP-based and describe the arithmetic operations that need to be calculated by the application as depicted in Figure 6. The execution result is displayed at both front-end component and client.

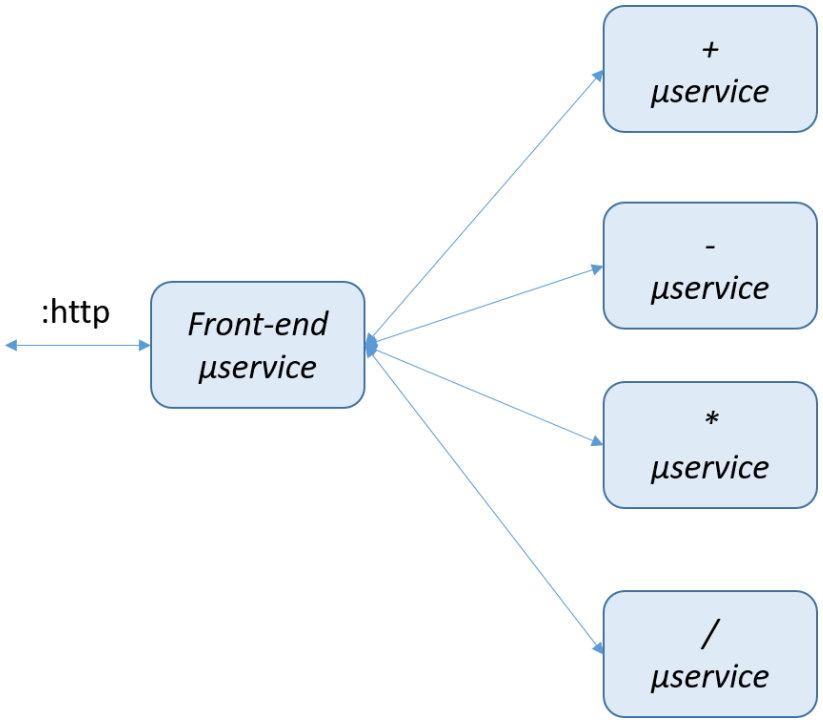


Figure 6 : Calculator service-based application

SumService: http://homepages.laas.fr/smedjiah/tmp/SumService.js
SubService: http://homepages.laas.fr/smedjiah/tmp/SubService.js
MulService: http://homepages.laas.fr/smedjiah/tmp/MulService.js
DivService: http://homepages.laas.fr/smedjiah/tmp/DivService.js
CalculatorService: http://homepages.laas.fr/smedjiah/tmp/CalculatorService.js

- To start a specific calculator µService

```
$ node <Service>.js
```

- But, before that, you need to install the required runtime (i.e. nodejs, npm) and cURL

```
$ sudo apt install nodejs
```

```
$sudo apt install npm
```

Ubuntu : `$ sudo apt install curl` or `#apt install curl`

Alpine : `$ sudo apk add curl` or `#apk add curl`

- To execute the services (using cURL)

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`

Part three: Deploy the Calculator application on OpenStack

With the newly installed OpenStack client:

- Create 5 VMs that will host the 5 µServices of the Calculator application and make sure you settle the right network connectivity of these VMs. For this step, you can use, either the Ubuntu image ***"Ubuntu4CLV"*** **(username = user/ password = user)** or the Alpine image ***"alpine-node"*** **(username = root / password = root)**

  Note that, the required runtime of the Calculator is already installed in the Alpine image. If the "alpine-node" image is not visible in the list of your proposed images, that means that it is not public and that you need to accept it before

- To list all available (and waiting for accepting images)

```
$ glance image-list --visibility shared --member-status pending
```

- To accept a specific image given its ID

```
$ openstack image set --accept %ID_DE_L'IMAGE%
```

- Start the services (see part one)
- Add the missing "sync-request" module and re start the services

```
$ curl -sL https://deb.nodesource.com/setup_16.x | sudo -E bash -
```

```
$ sudo apt install nodejs
```

```
$ npm install sync-request
```

- Test the execution using the appropriate HTTP requests (see part one)
- Using a text editor (e.g. nano), and without stopping the application, change the source code of one µService (e.g. adding some display instructions in between coding lines).
- Save and restart the modified µService
- Re-execute the application. What do you observe? Write down your comments.

Part four: Automate/Orchestrate the application deployment (optional)

- Re-deploy a re-execute the same application using a set of Docker nodes (you can re use the VirtualBox VMs that have already Docker installed on it). To that end, you need to use a Docker client. The list of proposed Docker clients is available via this link. For this task , we propose to use the NodeJS client. The installation and user guide of this client is available via this link.

  Note that you need to execute the js scripts with sudo

  ```
  $ sudo node <script>.js
  ```

4. Expected work for objectives 10 and 11

In this step, a Network as-a-Service (NaaS) provider proposes to process a prospective client request and deliver on-demand virtualized network topologies that include virtual hosts, routers, and communication functions implementing network levels from 3 to 7.

Part one: The client requirements and target network 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 sub-networks that could be described in what follows:

- Sub-Network 1 (IP = 192.168.1.0/24)
  - 1 virtual machine hosting the calculator front end service
  - 2 network appliances
    - § 1 router (IP 192.168.1.254) that provides access to end users
    - § 1 router (IP 192.168.1.253) that makes the bridge between the public network and sub-network 1
- Sub-Network 2 (IP = 192.168.2.0/24)

- 4 virtual machines that host the services implementing the four arithmetic operations
- 1 network appliance
  - 1 router (192.168.2.254) that makes the bridge between the 2 sub-networks

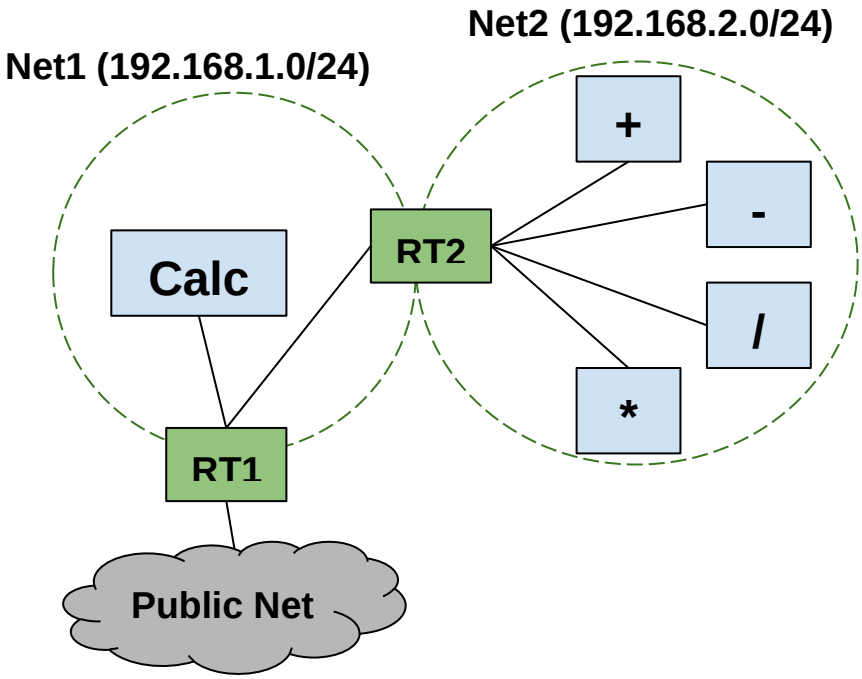


Figure 7 : Target network topology

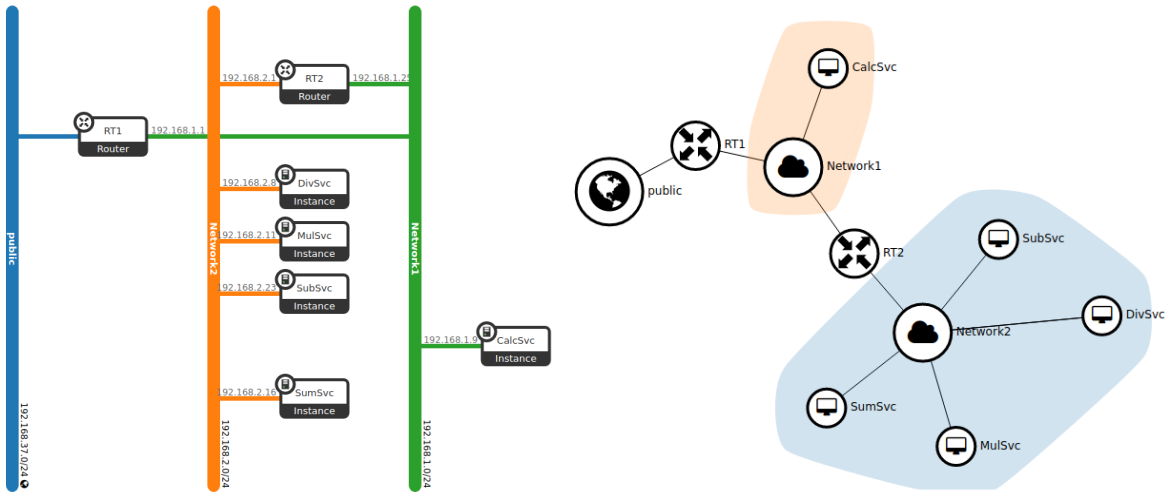


Figure 8 : Implementation of the target topology on OpenStack

Part two: Deployment of the target topology

- Prepare the 2 sub-networks according to the client request

- Instantiate the router to make up the topology described in Figures 7 and 8
- Instantiate the VMs that will host and execute the services. You should use “alpine-node” VMs
- Deploy the node.js services over them

Part three: Configuration of the topology and execution of the services

- 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?
- To fix the observed issue, you need to define a traffic route (from the calculator front end) between the 2 sub-networks

```
$ sudo route add -net @reseau_dest/mask gw passerelle
```

For example: `route add -net 192.168.0.0/masque gw 10.0.0.254`

- Test the the connectivity between the VMs
- Test the execution of the calculator using cURL like you already did in the previous section.