5e année ISS 2019/20

Cloud Computing: Adaptability and Autonomic Management

Lab 1 : Introduction to Cloud Hypervisors

Johan Alberti - Romain Saboret

Introduction

The general purpose of this lab is to enhance our 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 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/*

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.

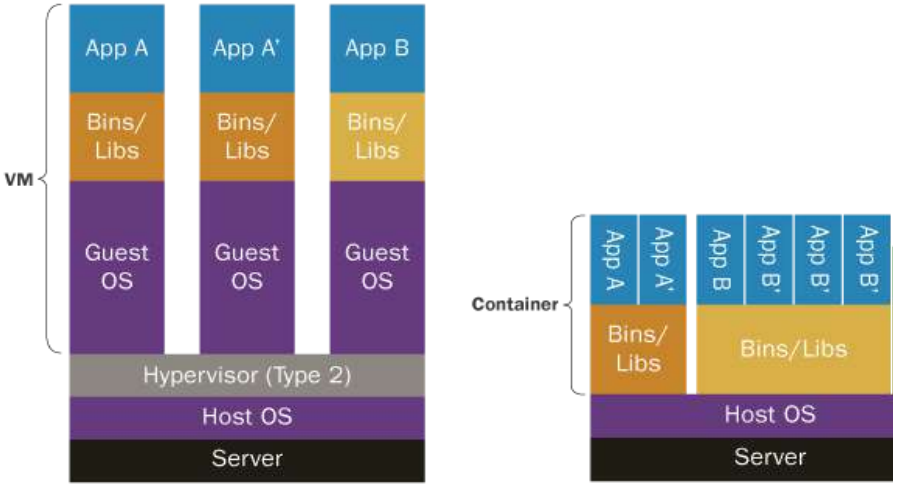


Figure 1: VM vs CT

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

- Understand the figure above, and elaborate on it.

Virtual machines and containers differ in several ways, but the primary difference is that containers provide a way to virtualize an OS so that multiple workloads can run on a single OS instance. With VMs, the hardware is being virtualized to run multiple OS instances. Containers' speed, agility, and portability make them yet another tool to help streamline software development.

- 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

Taking into account memory size and CPU usage, the container is better as it will load into memory only the chosen applications and not OSs plus applications as the VMs do.
For applications usage, both are the same for the hardware: the cost of memory size and CPU usage for the application is the same (but the container stays better as its virtualization is less resourceful).
For security, container can be broken up and let hackers into the server it runs on. VMs can not. VMs has separate resources with the original resources inaccessible.
For response time, container are quicker as VMs try to run OSs on small memory size.
Tools are available for containers to upgrade the applications' possibilities more easily than changing the whole OS on a VM.

No solution is perfect then, for an administrator's point of view, the VMs may seem better: sacrificing memory and available CPU usage to keep the maximum secured network can be better. For a developer's point of view, containers are way better as they work for all systems, use low resources and can be easily upgraded.

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.

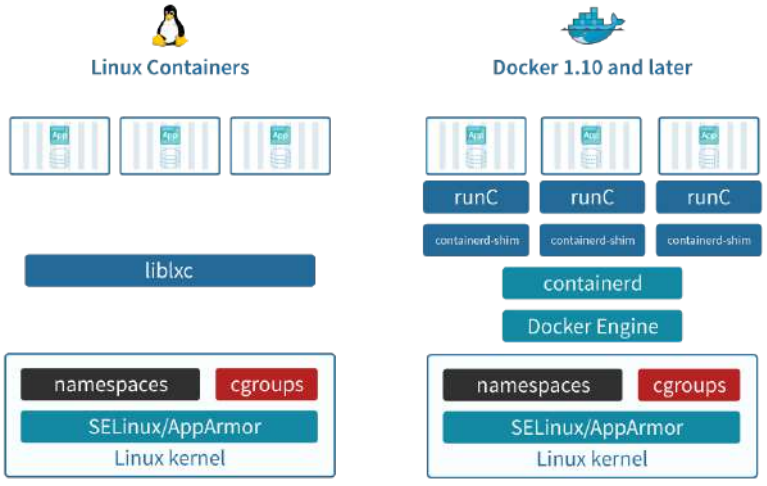


Figure 2 : Linux Lxc vs Docker

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,
- Application isolation and resources : Isolation is an essential criteria in terms of

security

- Containerization level : It characterizes the nature of what will be containerized. It can act from a wide container at the system level for several applications, or a multitude of smaller containers centered around each application.
- Tooling : This criteria characterizes the functions available to developers and administrators.

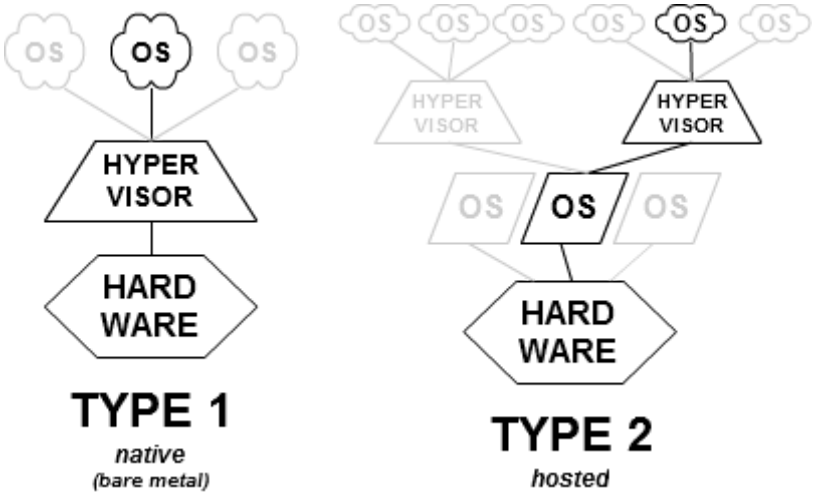
- By doing your own research on the Web, classify the existing CT technologies (LXC, Docker, Rocket, …) based on these criteria, and eventually, on additional criteria that you need to identify by yourself.

	LXC	Docker	Rocket
Isolation	No	Yes	No
Containerization level	Composable containers	Independant containers	Composable containers
Tooling	Medium	High	Low

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.

The type 1 hypervisors are built directly on the hardware (bare-metal) and can host different OS on it. These are not the common one for the average person.
Type 2 hypervisors are the usually known as Virtual Box and others. They look like applications and run on an OS to host OSs in them.

- Identify to which architecture type VirtualBox and OpenStack belong to.

VirtualBox and OpenStack are type 2 as they run on OSs (Windows and Linux architecture are the most common).

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 another 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.

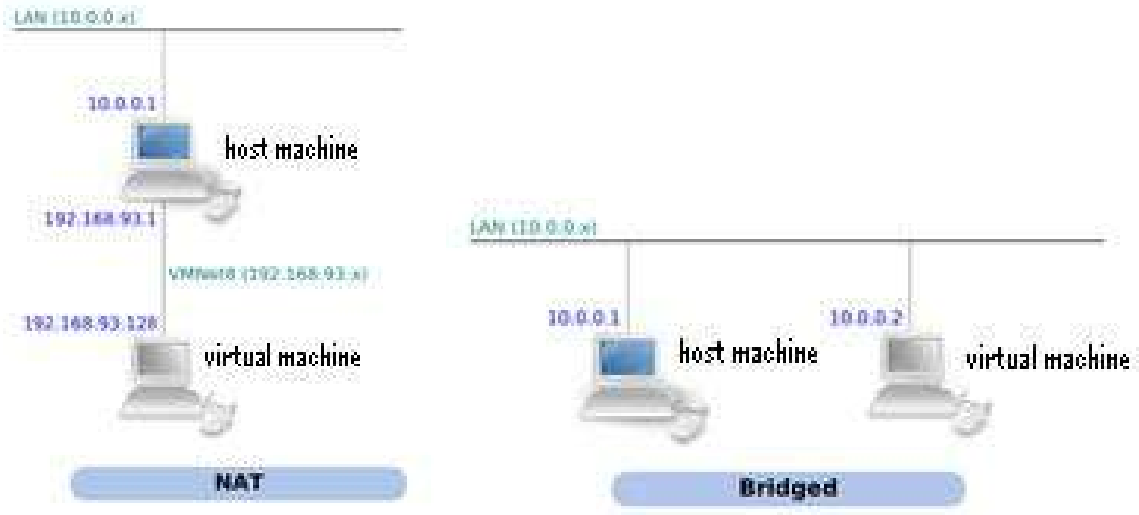


Figure 3 : Mode NAT vs Mode Bridge (the most usual)

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).

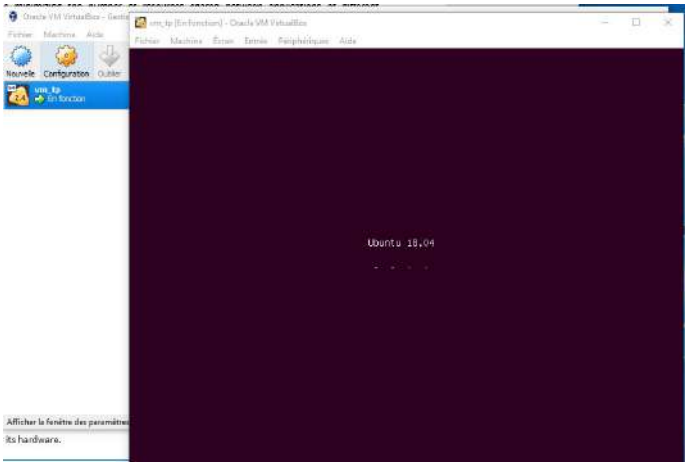
1. Tasks related to objectives 4 and 5

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 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).

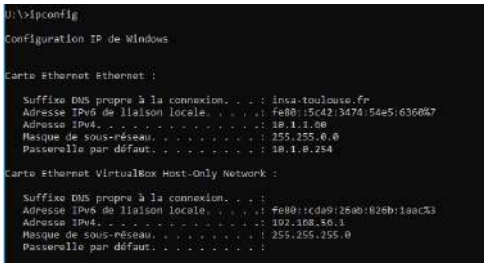
Tasks:

First part: Creating and configuring a VM

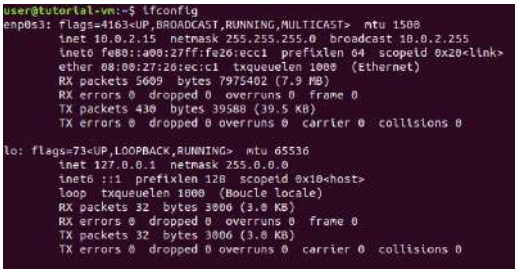


Second part: Testing the VM connectivity

IP address can be found with ifconfig (ipconfig) :



```
U:\>ipconfig

Configuration IP de Windows


Carte Ethernet Ethernet :

   Suffixe DNS propre à la connexion. . . : insa-toulouse.fr
   Adresse IPv6 de liaison locale. . . . .: fe80::5c42:3474:54e5:6360%7
   Adresse IPv4. . . . . . . . . . . . . .: 10.1.1.00
   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::cda9:2b0b:826b:1aac%3
   Adresse IPv4. . . . . . . . . . . . . .: 192.168.56.1
   Masque de sous-réseau. . . . . . . . . : 255.255.255.0
   Passerelle par défaut. . . . . . . . . :
```



```
user@tutorial-vm:~$ ifconfig
enp0s3: flags=4163<UP,BROADCAST,RUNNING,MULTICAST>  mtu 1500
        inet 10.0.2.15  netmask 255.255.255.0  broadcast 10.0.2.255
        inet6 fe80::a00:27ff:fe26:ecc1  prefixlen 64  scopeid 0x20<link>
        ether 08:00:27:26:ec:c1  txqueuelen 1000  (Ethernet)
        RX packets 5609  bytes 7975402 (7.9 MB)
        RX errors 0  dropped 0  overruns 0  frame 0
        TX packets 430  bytes 39588 (39.5 KB)
        TX errors 0  dropped 0 overruns 0  carrier 0  collisions 0

lo: flags=73<UP,LOOPBACK,RUNNING>  mtu 65536
        inet 127.0.0.1  netmask 255.0.0.0
        inet6 ::1  prefixlen 128  scopeid 0x10<host>
        loop  txqueuelen 1000  (Boucle locale)
        RX packets 32  bytes 3000 (3.0 KB)
        RX errors 0  dropped 0  overruns 0  frame 0
        TX packets 32  bytes 3000 (3.0 KB)
        TX errors 0  dropped 0 overruns 0  carrier 0  collisions 0
```

Let's summarize :

Host Windows PC	10.1.1.60
Carte Réseau de VirtualBox (NAT)	192.168.56.1
VM Linux 64 bits	10.0.2.15

Ping command :

- We use the following command “ping google.com” and got this result :

```
user@tutorial-vm:~$ ping google.com
PING google.com (172.217.19.142) 56(84) bytes of data.
64 bytes from par03s12-in-f142.1e100.net (172.217.19.142): icmp_seq=1 ttl=51 time=8.40 ms
64 bytes from par03s12-in-f142.1e100.net (172.217.19.142): icmp_seq=2 ttl=51 time=7.87 ms
64 bytes from par03s12-in-f142.1e100.net (172.217.19.142): icmp_seq=3 ttl=51 time=8.54 ms
64 bytes from par03s12-in-f142.1e100.net (172.217.19.142): icmp_seq=4 ttl=51 time=7.81 ms
^C
--- google.com ping statistics ---
4 packets transmitted, 4 received, 0% packet loss, time 3005ms
rtt min/avg/max/mdev = 7.816/8.160/8.546/0.317 ms
```

  So we have internet access from the VM.

- We try to reach the VM Linux from the Host Windows and got this :

```
U:\>ping 10.0.2.15

Envoi d’une requête 'Ping'  10.0.2.15 avec 32 octets de données :
Délai d’attente de la demande dépassé.

Statistiques Ping pour 10.0.2.15:
    Paquets : envoyés = 1, reçus = 0, perdus = 1 (perte 100%),
```

  Ping didn’t work so we cannot access the VM from the Host while the opposite is working.

  SSH is not working in both ways, we need to change some parameters.

```
user@tutorial-vm:~$ ssh 10.1.1.60
ssh: connect to host 10.1.1.60 port 22: Connection refused
```

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.

Nom	Protocole	IP hôte	Port hôte	IP invité	Port invité
ssh	TCP	10.1.1.60	22	10.0.2.15	22

Figure 4 : Port forwarding

Let’s try the previous test with SSH now :

```
user@tutorial-vm:~$ ssh 10.1.1.60
user@10.1.1.60's password:
```

Fourth part: VM duplication

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

- Run the following command:

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

    - *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 vm-alpine-copy.vmdk file

NB: The original copy of the file is not sufficient , because it embeds the same identifier as the original, which must be unique in the hypervisors namespace. VirtualBox will report an error.

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.
Use your already well-configured Virtualbox VM and execute the following steps:

1. Get an ubuntu image:

   ```
   $ sudo docker pull ubuntu
   ```

2. Execute an instance of the ubuntu image (CT1):

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

3. Install the required connectivity testing tools :

   ```
   $ sudo 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.

    The Docker IP adress is 172.17.0.2

    2. Ping an Internet resource from Docker

    We try to reach Google server and it works :

```
root@fe5fb7602c12:/# ping google.com
PING google.com (172.217.171.238) 56(84) bytes of data.
64 bytes from mrs09s07-in-f14.1e100.net (172.217.171.238): icmp_seq=1 ttl=50 time=8.14 ms
64 bytes from mrs09s07-in-f14.1e100.net (172.217.171.238): icmp_seq=2 ttl=50 time=7.35 ms
64 bytes from mrs09s07-in-f14.1e100.net (172.217.171.238): icmp_seq=3 ttl=50 time=8.30 ms
64 bytes from mrs09s07-in-f14.1e100.net (172.217.171.238): icmp_seq=4 ttl=50 time=7.34 ms
^C
--- google.com ping statistics ---
4 packets transmitted, 4 received, 0% packet loss, time 3004ms
rtt min/avg/max/mdev = 7.346/7.787/8.300/0.440 ms
```

3. Ping the VM from Docker

VM from Docker working also :

```
root@fe5fb7602c12:/# 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=0.039 ms
64 bytes from 10.0.2.15: icmp_seq=2 ttl=64 time=0.094 ms
64 bytes from 10.0.2.15: icmp_seq=3 ttl=64 time=0.096 ms
64 bytes from 10.0.2.15: icmp_seq=4 ttl=64 time=0.052 ms
64 bytes from 10.0.2.15: icmp_seq=5 ttl=64 time=0.070 ms
^C
--- 10.0.2.15 ping statistics ---
5 packets transmitted, 5 received, 0% packet loss, time 4095ms
rtt min/avg/max/mdev = 0.039/0.070/0.096/0.023 ms
```

4. Ping the Docker from the VM

Docker from VM also working

```
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=0.038 ms
64 bytes from 172.17.0.2: icmp_seq=2 ttl=64 time=0.079 ms
64 bytes from 172.17.0.2: icmp_seq=3 ttl=64 time=0.086 ms
64 bytes from 172.17.0.2: icmp_seq=4 ttl=64 time=0.052 ms
^C
--- 172.17.0.2 ping statistics ---
4 packets transmitted, 4 received, 0% packet loss, time 3070ms
rtt min/avg/max/mdev = 0.038/0.063/0.086/0.021 ms
```

Elaborate on the obtained results

From a container, we have access to the rest of the network and the network have access to the inside of the container.

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

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

6. Install nano (text editor) on CT2.

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

7. 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.

```
user@tutorial-vm:~$ sudo docker commit f11f76b9da01 test:test
sha256:3b2e3c2cb53f36e01d7dc5054d9c3d1026eb92685fe23abf9f8dc4206f7d7585
```

8. Stop and terminate CT2

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

```
user@tutorial-vm:~$ sudo docker stop f11f76b9da01
f11f76b9da01
user@tutorial-vm:~$ sudo docker rm f11f76b9da01
f11f76b9da01
```

9. List the available Docker images in the VM

```
$ sudo docker images
```

```
user@tutorial-vm:~$ sudo docker images
REPOSITORY          TAG                 IMAGE ID            CREATED             SIZE
test                test                3b2e3c2cb53f        3 minutes ago       93.1MB
ubuntu              latest              549b9b86cb8d        3 weeks ago         64.2MB
```

10. Execute a new instance (CT3) from the snapshot that you previously created from CT2 using the run command
11. `$ sudo docker run --name ct3 -it %REPO:%TAG`

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

```
  GNU nano 2.9.3   Nouvel espace   Modifié

S



^G Aide     ^O Écrire   ^W Chercher ^K Couper
^X Quitter  ^R Lire fic ^\ Remplace ^U Coller
```

    Yes we still have nano, in fact taking a snapshot of a CT enables us to restore it to its state later. For example, if we CT then uninstall nano, and then restore it to the snapshot, nano will be back.

11. Docker enables "recipes" sharing to create persistent images (as a second alternatives 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 .
```

```
user@tutorial-vm:~$ sudo docker images
REPOSITORY          TAG                 IMAGE ID            CREATED             SIZE
test2               test2               9ee0aefcdbd3        2 minutes ago       93.1MB
test                test                3b2e3c2cb53f        18 minutes ago      93.1MB
ubuntu              latest              549b9b86cb8d        3 weeks ago         64.2MB
```

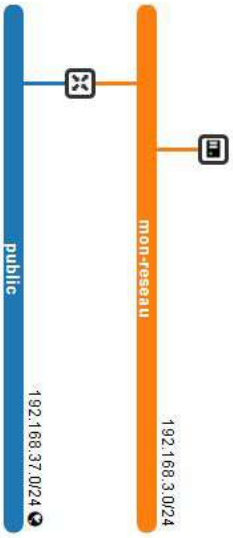
2. Expected work for objectives 6 and 7

Tasks :

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 "**INSAT**" 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 alpine image or, possibly, the one that you did export from VirtualBox. The VM will automatically started and all related information are displayed in the dashboard. What do you observe?

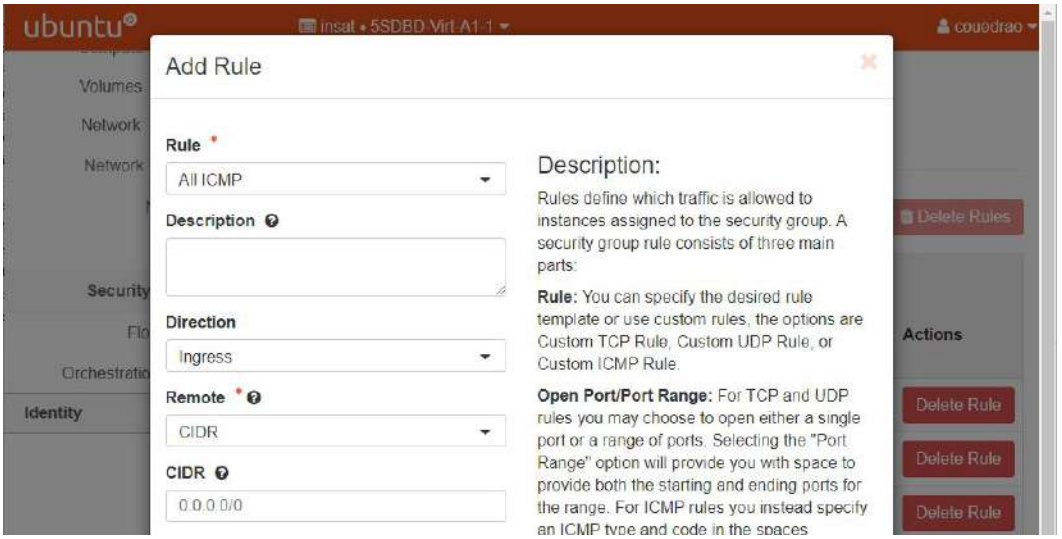
We have created a private network and a router between the private and the public networks. Then we created an instance on the private network.

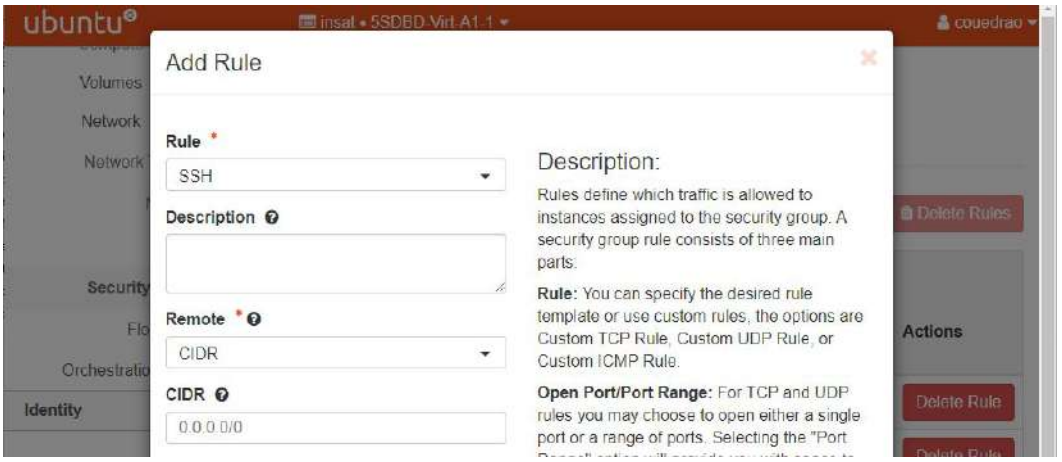


Configuring the security rules in OpenStack

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/authorise 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? It's a consistent adress

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

Not working

```
$ ping 192.168.56.1
PING 192.168.56.1 (192.168.56.1): 56 data bytes
^C
--- 192.168.56.1 ping statistics ---
70 packets transmitted, 0 packets received, 100% packet loss
```

  - Check the connectivity from the desktop to the VM. Write down your comments.

Not working

```
U:\>ping 192.168.3.15

Envoi d'une requête 'Ping'  192.168.3.15 avec 32 octets de données :
Délai d'attente de la demande dépassé.

Statistiques Ping pour 192.168.3.15:
    Paquets : envoyés = 1, reçus = 0, perdus = 1 (perte 100%),
```

- 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 routage 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 b 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.

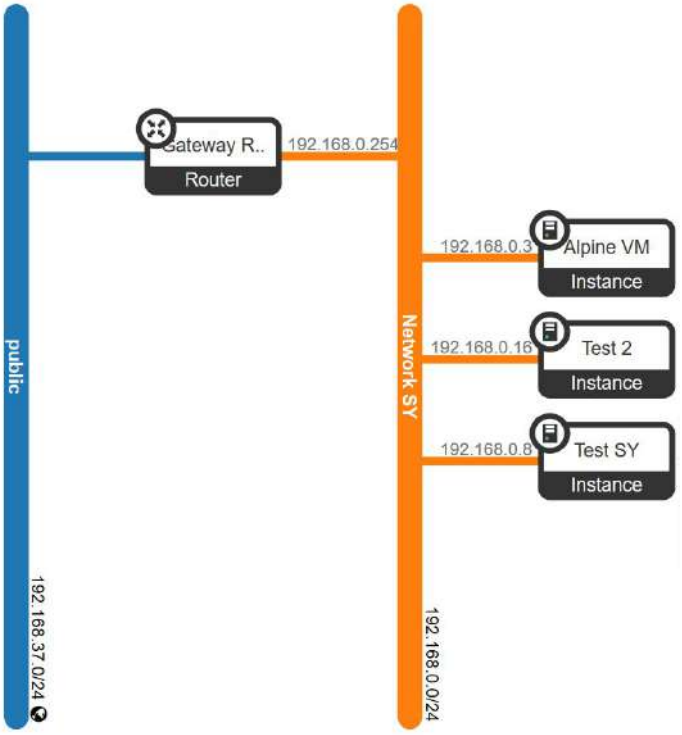


Figure 5 : Target Network Topology

- 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.

  SSH works from outside.

```
U:\>ssh cirros@192.168.37.77
cirros@192.168.37.77's password:
$
```

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

  When we choose resize, the Task become “Resizing or Migrating” then the status change to “Confirm resize” then we can resize. The VM never stop running.

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

  The exact same thing happens. The VM stays shut down.

- 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 do highlight an existing technical limitation (related to OpenStack). What is it? Can you propose a hint/way to address it?

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

Expected work for objectives 8 and S9

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**

```
$ 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
```

```
user@tutorial-vm:~$ openstack
(openstack) help

Shell commands (type help <topic>):
===================================
cmdenvironment  exit  history  py        quit  save  shell      show
edit            help  load     pyscript  run   set   shortcuts

Application commands (type help <topic>):
=========================================
access token create                    network rbac list
address scope create                   network rbac set
address scope delete                   network rbac show
address scope list                     network segment create
address scope set                      network segment delete
address scope show                     network segment list
aggregate add host                     network segment set
```

- 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 an NodeJs micro-service. In addition, there is a fifth micro-service that implement 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.

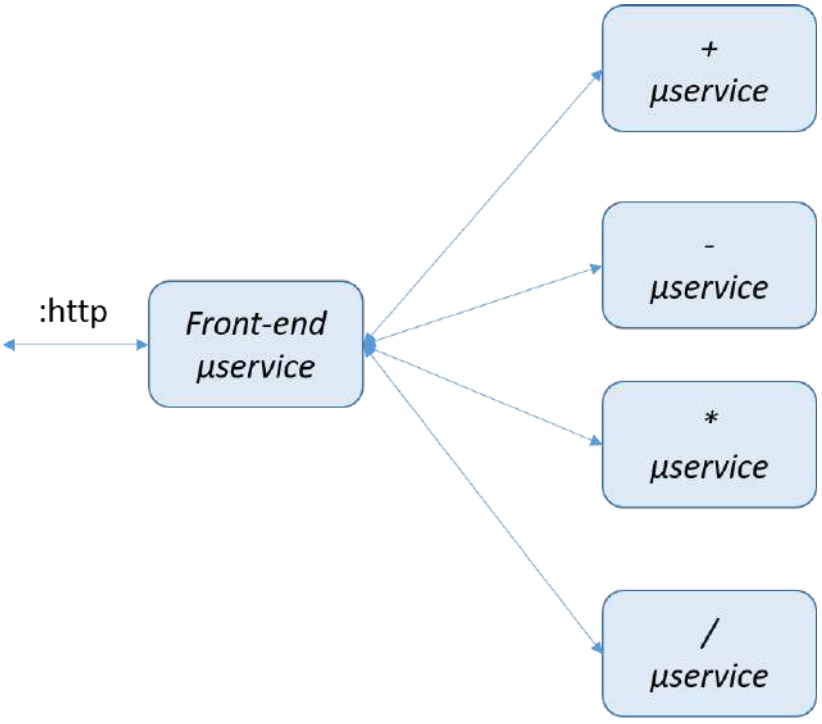


Figure 6 : Calculator service-based application

The µServices archives are available in the following **Shared Folder**

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`

Let's see an example :

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

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, 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)
- Test the execution using the appropriate HTTP requests (see part one)

We put this command in the ubuntu local VM :

```
user@tutorial-vm:~$ curl -d "12+5" -X POST http://192.168.37.77:80
result = 17
```

And the master µservice is listening on port 80 so it redirect to the slave uservice :

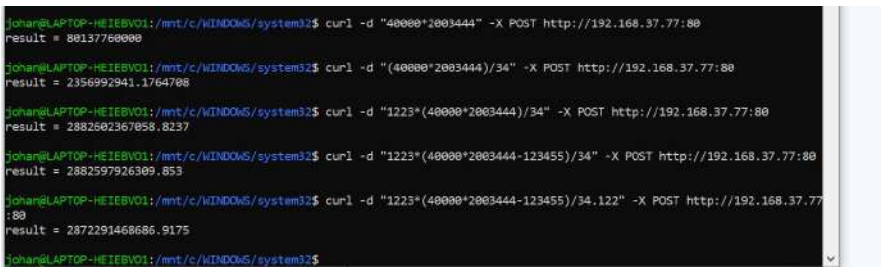
```
alpine-node:~# node CalculatorService.js
Listening on port : 80
New request :
12+5 = 17
```

The slave µservice is listening on port 50001 on another VM and is called by the master µservice.

```
alpine-node:~# node SumService.js


Listening on port : 50001
New request :
A = 12
B = 5
A + B = 17
```

And it’s why we got the result in the first shell.

`From another pc connected with WIFI on INSA network :`



- Using a text editor (e.g. nano), and without stopping the application, change the source 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.

```
alpine-node:~# node CalculatorService.js
Listening on port : 80
hello.
_
```

It works as expected:

Part four: Automate/Orchestrate the application deployment

- Re deploy and re execute the same application using a set of Docker nodes (you can re use the VirtualBox VMs that has already Docker installed on it). To that end, you need to use a Docker client. The list of proposed Docker client 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
```

We created a snapshot of an image based of ubuntu containing all the necessary services and programs (npm, nodejs, wget, nano).

```
user@tutorial-vm:~$ sudo docker images
REPOSITORY          TAG                 IMAGE ID            CREATED             SIZE
services            all                 359489338d86        26 seconds ago      405MB
```

We run 5 times this images and redirect the requests between every containers.

4. Expected work for objectives 10 and 11

In this step, as NaaS (Network as-a-Service) provider, we propose 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 a 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

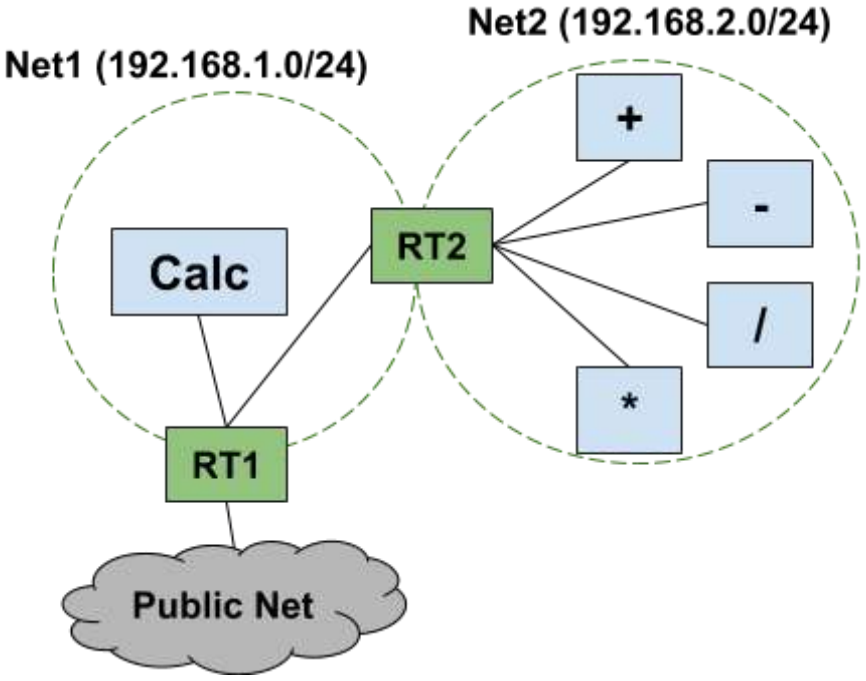


Figure 7 : Target network topology

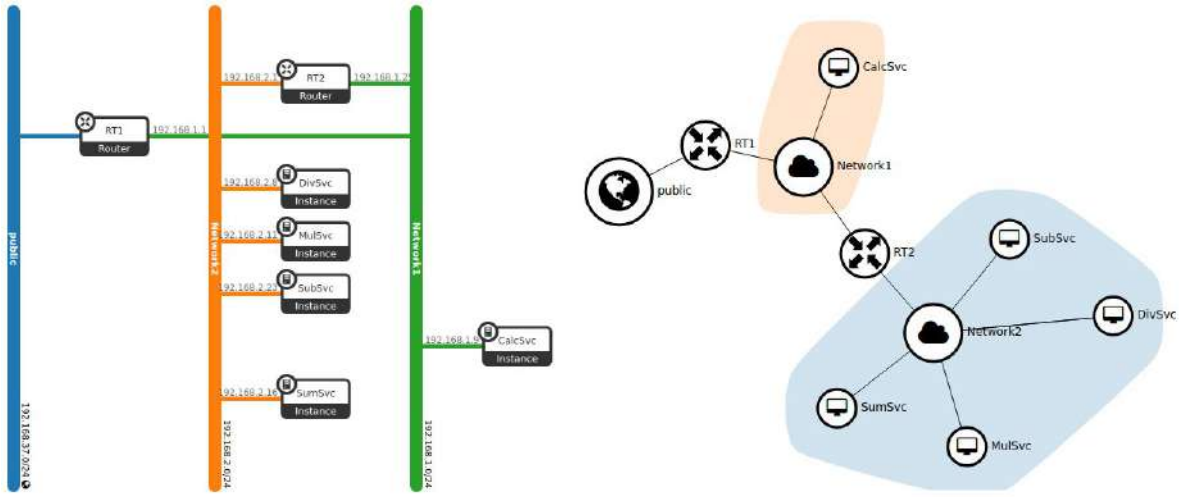


Figure 8 : Implementation of the target topology on OpenStack

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 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?

It’s not working, we got “connection refused”, we cannot access the subservice.

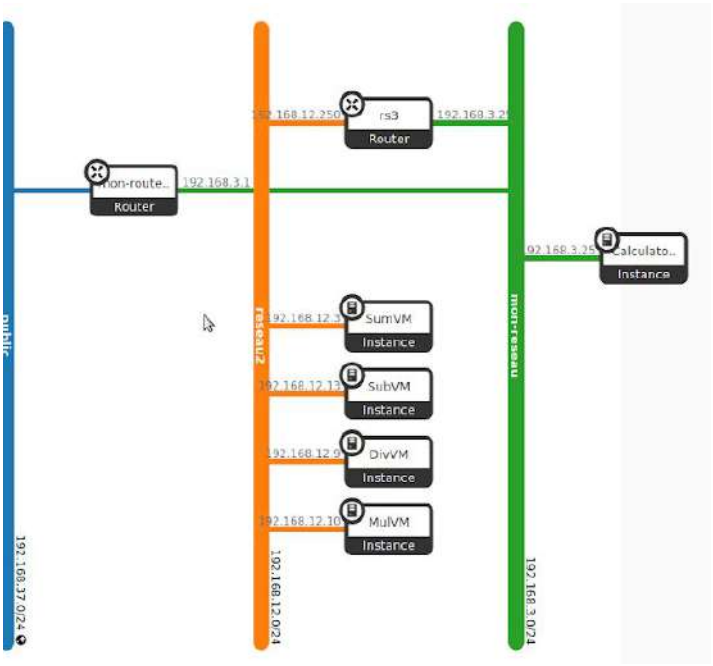
- To fix the observed issue, you need to define a traffic route (from the calculator front end) between the 2 sub-networks

```
$ 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.

Finally, we got this topology :



From a distant PC connected on INSA's WIFI network, we run a calculation with curl :

```
johan@LAPTOP-HEIEBVO1:/mnt/c$ curl -d "3*5" -X POST http://192.168.37.77:80
result = 15

johan@LAPTOP-HEIEBVO1:/mnt/c$
```

And we can see the result appearing almost directly on the distant PC while on serveur we have :

```
alpine-node:~# node CalculatorService.js
Listening on port : 80
hello.
New request :
3*4 = 12


New request :
3*5 = 15
```

```
alpine-node:~# node MulService.js
Listening on port : 50003
New request :
A = 3
B = 4
A * B = 12


New request :
A = 3
B = 5
A * B = 15
```

5e année ISS 2019/20

Cloud Computing: Adaptability and Autonomic Management

Lab 2: Provisioning End-user Application in Cloud Platforms

S. Yangui (INSA/LAAS)

Accès au sujet : tiny.cc/TP_Cloud2

Objectives

This lab aims at given students a good understanding of the service-based application lifecycle management in cloud computing context. The students will study the several provisioning process phases and investigate the way they should be implemented when provisioning: (i) sample HelloWorld code on existing and various cloud (PaaS-level) platforms (i.e. Google Cloud Platform, Cloud Foundry and Jelastic) and, (ii) the Resources Generator and Manager the students developed as part as Lab 2. The detailed objectives are detailed in what follows:

- Objective 1: Read and assimilate the several phases that make up cloud end-user applications provisioning process.
- Objective 2: Implement such a process using sample JEE application on Google Cloud Platform, Cloud Foundry and Jelastic. Various approaches and tools will be used for each one of the previously mentioned PaaS (e.g. through a plugin, a CLI and the Web).
- Objective 3: Learn how to build a more complex cloud application, and the application architecture concepts/principles you will need to succeed.
- Objective 4: Implement the lessons learned from Objective 3 to design and develop a simple MVC application with database access, on a PaaS.
- Objective 5: Design and develop your own embryonic PaaS using LXC containers on top of the Proxmox infrastructure and migrate your Resources Generator and Manager applications over there.

Provided Materials:

- Text addressing Objectives 1 and 3 that you need to read very carefully, several times if necessary.
- For Cloud Foundry : Sample JEE application artifacts (HelloWorld Web archives)
- For Google App Engine: Set of valid credentials

Email	Password

5iss.insa2020@gmail.com	5Iss.2020!
yy3155339A@gmail.com	5iss_A!?
gei5iss.a2@gmail.com	5GEIISSa2
yy3155339C@gmail.com	5iss_C!?
yy3155339D@gmail.com	5iss_D!?

Note: For Jelastic and Cloud Foundry, you need to register and create your own credentials by yourself. It is easy, fast and free.

- For Google App Engine : Required documentation
  - Environment setup (using gcloud)
  - Environment setup (using Eclipse)

Required Release:

- HelloWorld Servlet provisioned on Google Cloud Platform, Cloud Foundry and Jelastic. You have to provide the Lab's teacher with the link so he can test the application execution on the target PaaS. You imperatively need to deliver this within the lab hours.

General notes:

- Clarification: In this lab, "cloud applications" refers to the "end-user applications".
- Terminology 1: "Cloud" applications are also known as "cloud-based", "cloud-ready" or "cloud-enabled" applications. "Cloud-friendly" means the same although it is much less used.
- Terminology 2: Provisioning cloud applications is a continuous loop process that implements the whole applications lifecycle (i.e. develop, deploy and manage). This includes then the design and build activities previously mentioned. More generally, the different activities related to each one of these three phases will be discussed in detail in the rest of this document.

Task 1. Addressing Objective 1: Read and assimilate the several phases and steps that make up cloud end-user applications provisioning process

Provisioning cloud resources process consists of three main phases. Such a process implements the resources lifecycle and is strongly inspired from Service-oriented Computing paradigm (SOC) and the Telecommunications Information Networking Architecture (TINA) [1] [2]. In particular, provisioning cloud (end-user) applications in PaaS according to such process is briefly discussed in [3]. In what follows, in-depth details of the different phases:

1. Phase 1: Development

It involves (non-exhaustive list, not all sub-phases are mandatory, some of them might be concurrent):

- Modeling the application (e.g. designing the required service / component-based topology)

- Developing the sources (e.g. writing code and scripts, referring to remote components/modules published over repositories, importing required libs)
- Compiling the sources
- Testing and debugging
- Building the artifacts (a.k.a executables, archives, deployables)
- Packaging in appropriate wrappers if necessary (e.g. service containers, standalone frameworks)
- Modeling the management plan
- Settling the SLAs
- Specifying the deployment descriptor (a.k.a manifest)

2. Phase 2: Deployment

It involves:

- Allocating the required cloud resources
- Uploading the executables over these resources
- Activating the application according to the specified plans/SLAs (including the automatic binding and activation of the required services – such as storage, logging, monitoring and so on.

3. Phase 3: Management

It involves:

- Executing the application
- Performing the appropriate/required management operations that aim at optimizing its performance and reducing its operation cost. SLAs violations, from end-user point of view, would trigger the execution of a specific management operation. Management operations are specified through the management plans and might be implemented as workflows. Scaling up/down or migrating an application component are among the examples of management operations.

Task 2. Addressing Objective 2: Implement part of the provisioning process using sample JEE application on Google Cloud Platform, Cloud Foundry and Jelastic

As part of this task, you need to provision a HelloWorld Servlet on:

- Google Cloud Platform using Eclipse and Google Plugin **OR** using google SDK (gcloud)
- Cloud Foundry using its CLI software and the Web console
- Jelastic using the Web console

Depending on the case study, you need to implement one or several steps for each phase of the application provisioning process.

Task 2.1 Provisioning HelloWorld Servlet on Google Cloud Platform

The goal of this subtask is to provision a HelloWorld Servlet on Google Cloud Platform. We will use Eclipse as IDE, as well as, the appropriate Google plugin.

For the development phase, you will need to:

- Install Google Plugin and Google SDK to settle your development environment
- Debug your code and “emulate” its deployment on the cloud (Google SDK allows this emulation on your own machine as if you were in the Cloud)

- Create a Google development project and link it to your local application

Note that for the development phase, there is no code to write. The sample code of the HelloWorld servlet is automatically generated by Google SDK.

For the deployment phase, you will push your application to the distant PaaS.

For the management phase, you will execute your application and get familiar with the Google management dashboard.

Work to do :

1. Install the Google Eclipse Plugin . Select **Help > Install New Software** in the bar menu and insert the following URL: https://dl.google.com/eclipse/plugin/4.6

Note: You may decide to use the novel tool launched by Google late 2018, i.e. Google Cloud Tools instead in order to proceed to the same deployment procedure.

Note: This plugin URL provided above is specific for Eclipse Kepler. For other distributions, you need to select the suitable plugin through the following link:

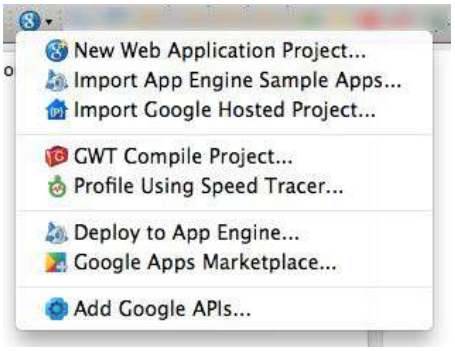
https://developers.google.com/eclipse/docs/install-eclipse-4.6

For the installation, check the followings plugins:

- Google Plugin for Eclipse
- SDK > Google App Engine Java SDK
- SDK > Google Web Toolkit SDK (optional).

Finally, you need to accept the review licenses and validate to start the installation. At the end, you need to validate the security warning and restart Eclipse.

If the installation is successful, a new Google icon with a specific menu will be added to your Eclipse.



Note: In order to support and develop GAE projects with Maven, please refer to: https://cloud.google.com/appengine/docs/java/tools/maven

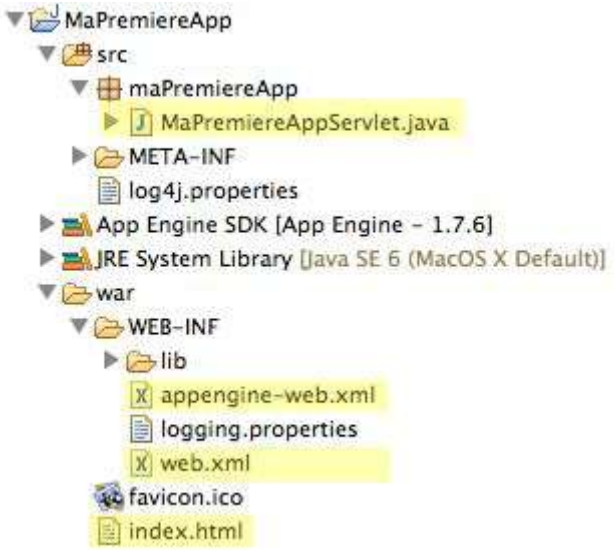
2. Create a new Google APP Engine application by clicking on the "**g**" Google button on Eclipse and select **New Web Application Project**.

The application creation wizard is displayed. You need to:

- Insert the project name (e.g. MyFirstApp)
- Insert the package name (e.g. myFirstApp)

- Uncheck “Use Google Web Toolkit” option.
- Check “Use Google App Engine” option

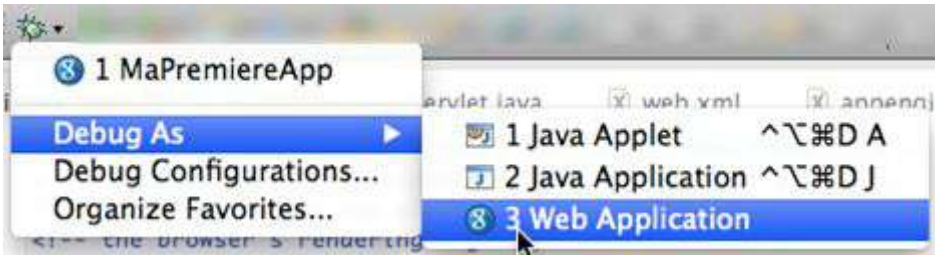
The created project architecture consists of a set of files and repositories. The content structure is very similar to a classical JEE project structure. The main files are highlighted in the following snapshot. Note that there is a specific XML file (i.e. appengine-web.xml) under \WEB-INF. Such a file allows to configure the use of Google resources/services (e.g. sessions management) by your application.



where :

- appengine-web.xml = google specification indicating the application identificator, version of the last code, ...
- web.xml = deployment descriptor of the application (application name, description, servlets JAVA classes, servlets mappings ⇔ “servlet-URL”)
- index.html = default welcome web page when the servlet is invoked.

3. Debug your application and test it locally (**Debug as" > "Web Application")**



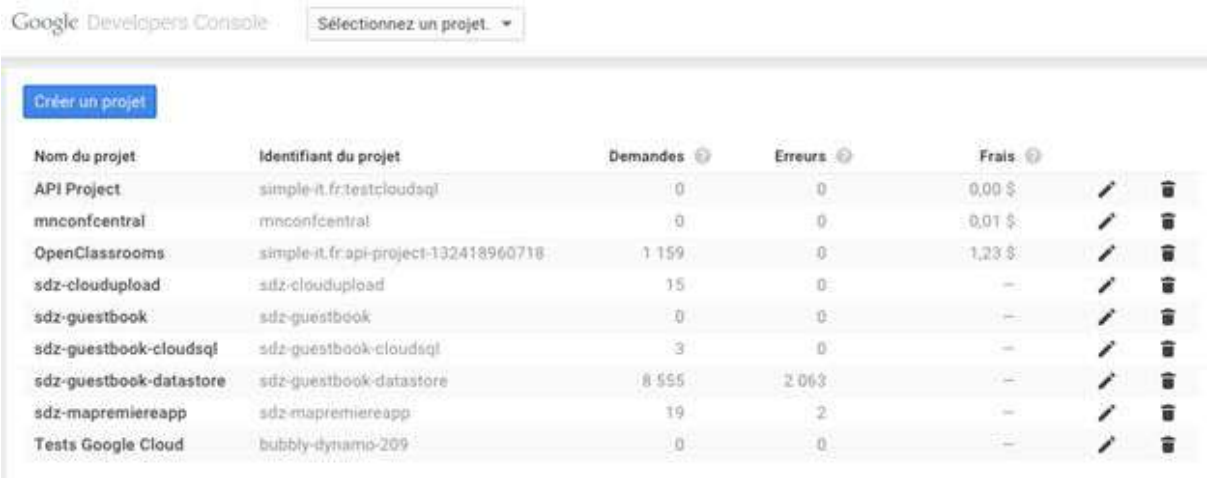
You should have **“*INFO: Dev App Server is now running*”** displayed then on your Eclipse console. This means that your local google mini-server is running.

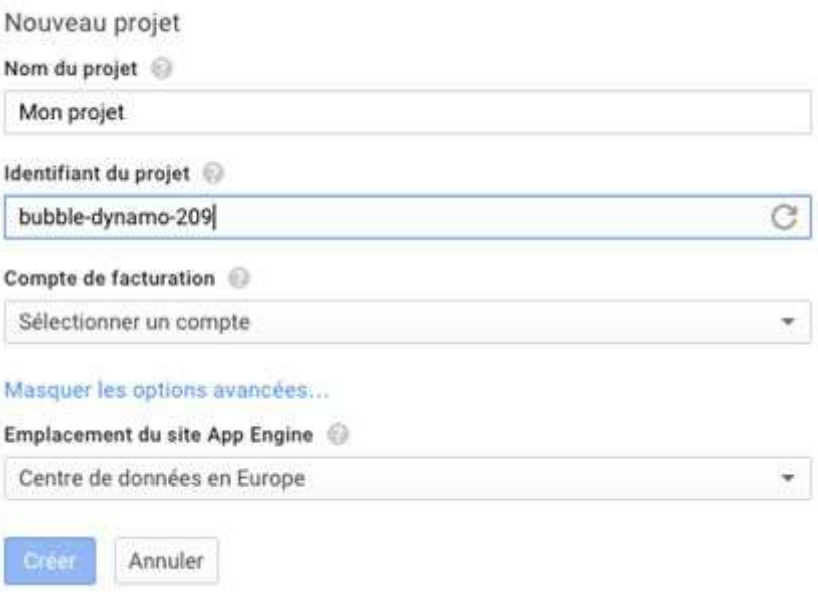
Use a browser and go to http://localhost:8888/ to test. If everything works well, you will be redirected to your index.html page.

Test the execution.

Note that it is possible to administer and monitor your application using the local administration tool provided by the plugin. The administration console is accessible at: http://localhost:8888/_ah/admin

4. Declare your application to Google. To that end, you need to visit http://console.developers.google.com, create a project and and get a unique ID for it.

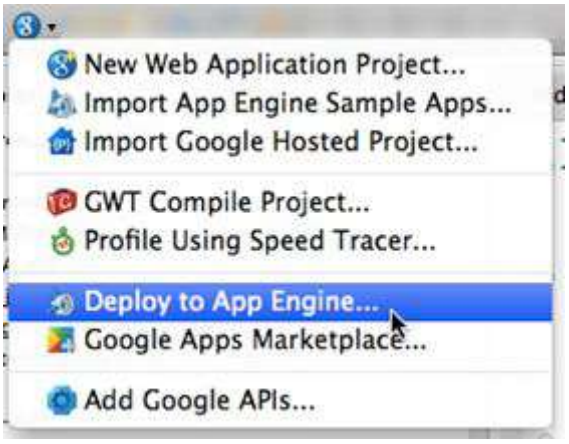




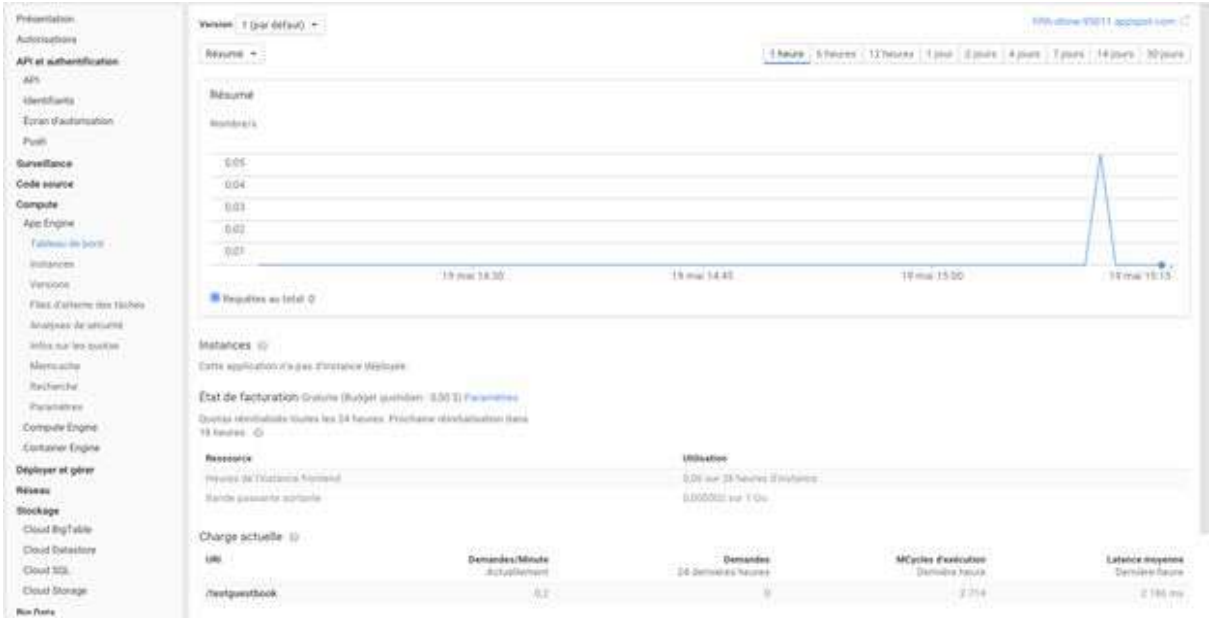
5. Configure your application and make it ready for deployment. Copy and paste this ID between the <*application*> elements in your **appengine-web.xml** file.

```
<?xml version="1.0" encoding="utf-8"?>

<appengine-web-app xmlns="http://appengine.google.com/ns/1.0">
<application>sdz-myfirstapp</application>
<version>1</version>
```

6. Deploy your application. Click on "**Deploy to App Engine**" in the google menu (Authentication to Google is required).



7. Manage your application. Once your App is deployed, you can manage it through the administration dashboard (Compute / App Engine / Dashbord).



Task 2.2 Provisioning HelloWorld Servlet on Cloud Foundry

The goal of this subtask is to provision a HelloWorld Sevlet on Cloud Foundry. To that end, we are going to use: (i) Pivotal Web Services (PWS) that provides Cloud Foundry as a Web service (deployed on top of AWS) and (ii) CF CLI.

For the development phase, we provide you with the application artifacts ready; however, you will need to:

- Install and configure the CF CLI as part of your development environment
- Settle the deployment descriptor (manifest.yml) provided with the artifact

For the deployment phase, you need to:

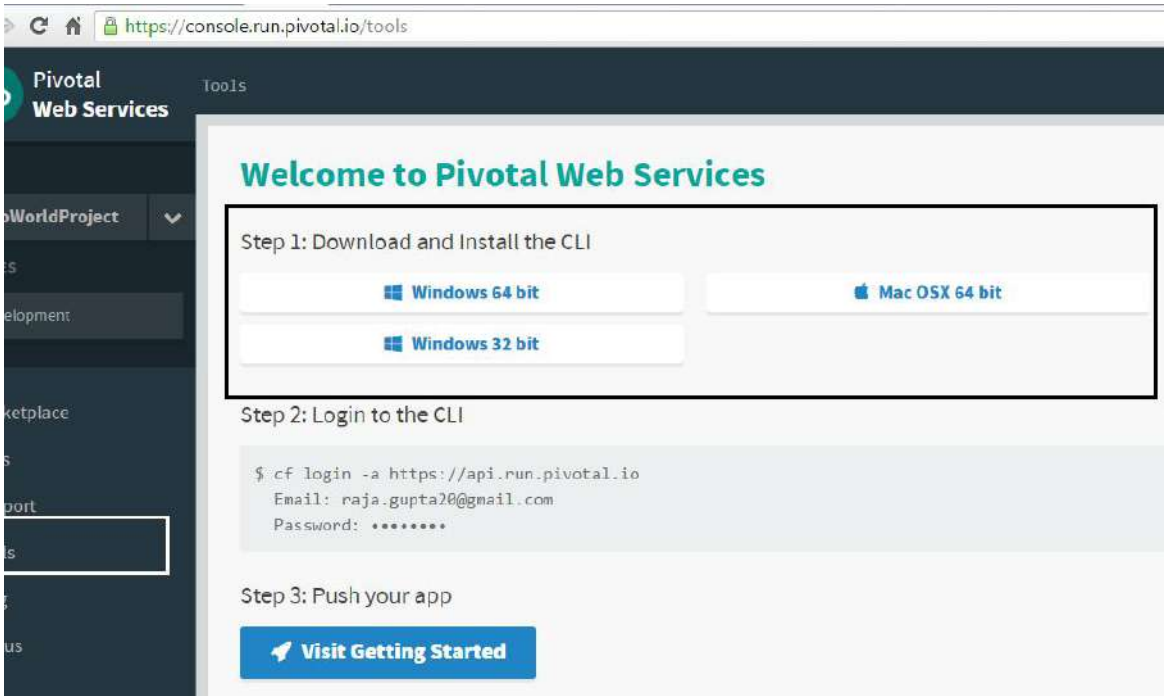
- Prepare the hosting environment on CF (creation of correspondent organization and space as target domain for your application)
- Push the artifacts there.

Note that for Cloud Foundry, the allocation of the PaaS resources is implicitly handled by the PaaS. Depending on the application type and implementation (i.e. JEE application in this case), the required PaaS resources (e.g. Apache Tomcat container and nginx router in this case) will be automatically allocated and configured by the PaaS rather than the developer.

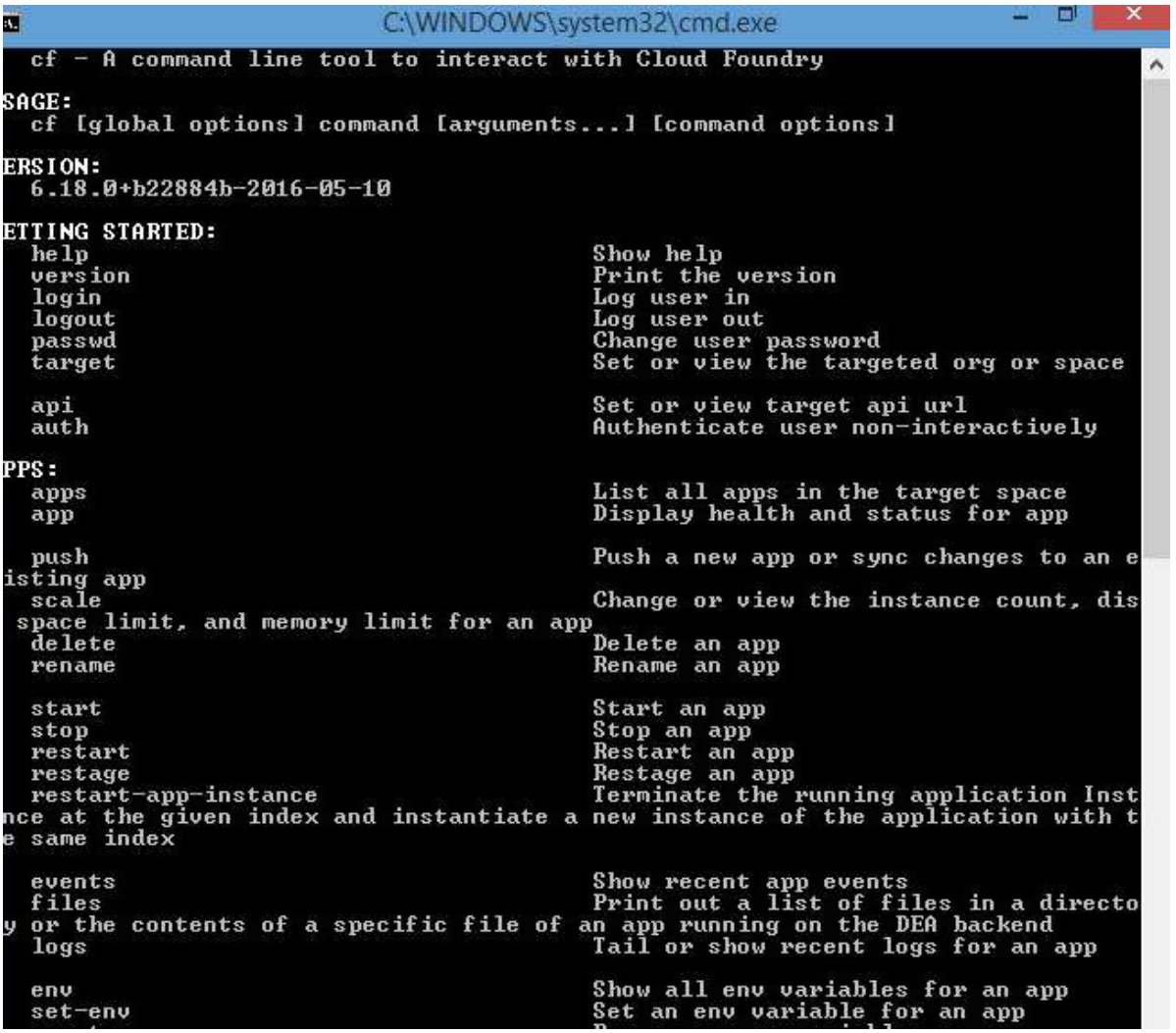
For the management phase, you will need to:

- Execute the application through the generated URL
- Scale up/down the application
- Bind it with a storage service
- Stop the application and terminate it

1. Connect to Pivotal Web Services , sign up and log in
2. Install the Cloud Foundry command line interface CF CLI (see http://docs.run.pivotal.io/cf-cli/install-go-cli.html).
   Tip: The simplest and the fastest way is to get the installer from the Web console (under Tools). If you are working on the INSA machines, you need to subvert the privileges restrictions when installing the client (See Appendix 1).



3. Open Command Prompt and run ***"cf help"*** to confirm that the tool is installed correctly. The example shows the beginning lines of output for this command. Read all the supported commands so you get familiar with the CLI capabilities. The comprehensive guidelines are available through the following link.
4. Enter command ***"cf api api.run.pivotal.io"*** to set the API endpoint. ***api.run.pivotal.io*** indicates the API of the public Cloud Foundry instances that we



are going to use and that is provided on top of Amazon infrastructure.
5. Enter command ***"cf login"*** to connect to the PaaS
6. Enter command ***"cf target [-o ORG] [-s SPACE]"*** to create and select a specific space under a given organization (e.g. WEB or INSA applications as ORG and DEV as space).
7. Download the HelloWorld artifacts and open the deployment descriptor (manifest.yml) to edit: (i) the application name (host attribute) and (ii) the number of the application's instances to be deployed (instances attribute).
8. Enter command ***"CD [HelloWorld_PATH]"*** to place your prompt under its folder.
9. Enter command ***"cf push"*** to upload the application artifact on cloud Foundry. The operation will not succeed and you will have errors. Based on the displayed information on your command line, can you explain this failure? For more details, you

might open and go through the logs using the appropriate commands or displaying them on the Web console.

```
2017-10-18T19:21:31.28+0200 [CELL/0] OUT Creating container
2017-10-18T19:21:31.80+0200 [CELL/0] OUT Successfully created container
2017-10-18T19:21:34.93+0200 [CELL/0] OUT Starting health monitoring of container
2017-10-18T19:21:35.05+0200 [APP/PROC/WEB/0] ERR Cannot calculate JVM memory configuration: There is insufficient mem
ry remaining for heap. Memory limit 256M is less than allocated memory 654457K (-XX:ReservedCodeCacheSize=240M, -XX:Max
irectMemorySize=10M, -XX:MaxMetaspaceSize=75931K, -XX:CompressedClassSpaceSize=15326K, -Xss1M * 300 threads)
2017-10-18T19:21:35.06+0200 [APP/PROC/WEB/0] OUT Exit status 1
2017-10-18T19:21:35.06+0200 [CELL/SSHD/0] OUT Exit status 0
2017-10-18T19:21:35.07+0200 [CELL/0] OUT Stopping instance a4bccbf9-6da5-46a0-5af3-2dae
2017-10-18T19:21:35.07+0200 [CELL/0] OUT Destroying container
```

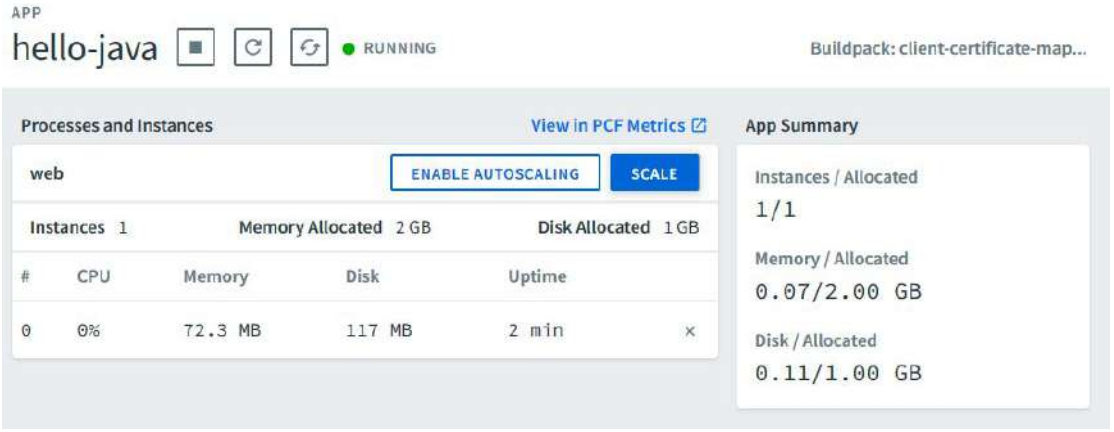
The host is taken, we need to change the host.

10. Fix the error and retry to start the application once again.

There was not enough allocated memory, we put 2 GB instead of 256 MB.

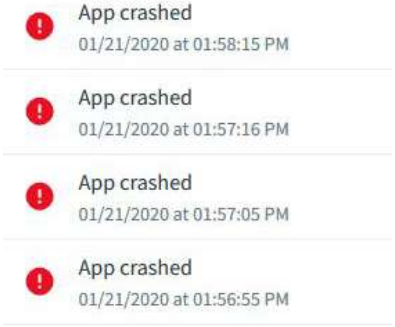
11. Execute the application one more time (via a browser)

12. Using the Web console, check the applications status, used resources and monitored performances.

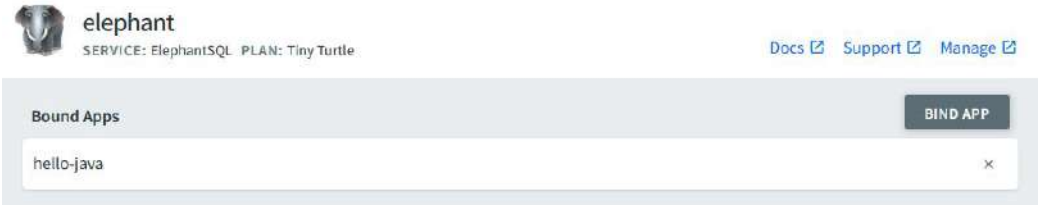


13. Scale up (respectively down) the application and try to investigate the impact of such operations on its behavior, performance, etc.
    Tip: Do not hesitate to execute it several times to that end.

Without enough memory, the server doesn't start.



14. Bind the application with a technical service from CF marketplace. There are few free services there that you can use such as basic SQL storage service.



We successfully bound the app with a SQL client.

Task 2.3 Provisioning HelloWorld Servlet on Jelastic

The goal of this subtask is to provision the same HelloWorld Servlet on Jelastic PaaS. To that end, we are going to use the public instance deployed over Hidora, the Swiss infrastructure provider accessible through the following link.

For the development phase, you will reuse the HelloWorld.WAR under (\target); however, you will need to settle the hosting environment of your application by yourself. Indeed, unlike CF, Jelastic requires manual and explicit settings of the required PaaS resources in terms routers, service containers, storage services and so on.

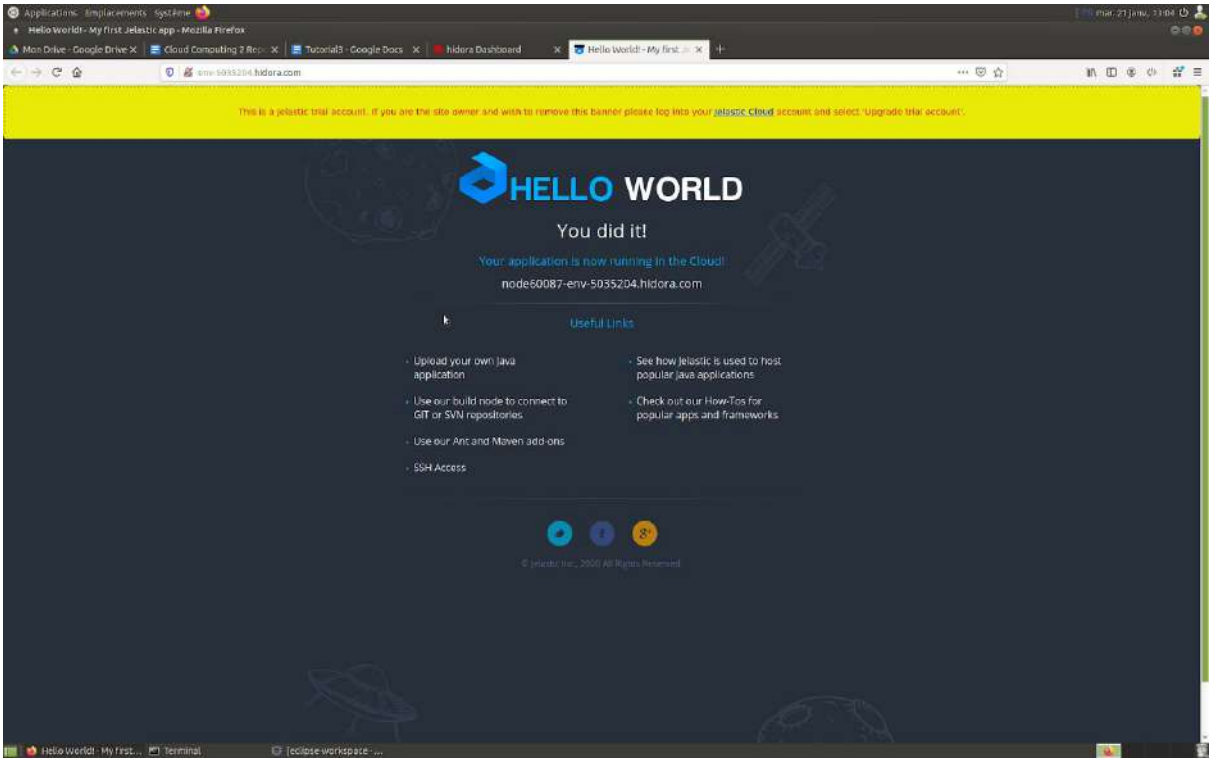
For the deployment phase, you will need to: (i) upload the application artifacts on Jelastic and, (ii) deploy it over the already created environment. Note that the concrete deployment in Jelastic is the binding between an application archives and a potential environment.

For the management phase, you will need to test the application execution.

1. Connect to Jelastic, select Hidora as IaaS provider and register to create a free trial account (Link)
2. Log in
3. Create the environment that you deem necessary to host and execute your HelloWorld Servlet

4. Upload your application artifacts

5. Deploy it
6. Execute it



Task 3 (Addressing Objective 3: Learn how to build a more complex cloud application, and the application architecture concepts/principles you will need to succeed)

Anyone who has built applications understands that applications that are specifically designed for the target platform will perform better and are more resilient and easier to manage. Designing applications for cloud platforms is no exception.

In the following, we discuss the key concepts to exactly understand how to go about designing and building optimal cloud applications. Note that simply migrating regular and legacy (complex) applications to the cloud as they are, although most of them are portable such as JEE applications, has led to poorly designed applications for cloud-based platforms, in the sense that they do not deliver the value on the cloud platform that enterprises expect.

Provisioning cloud applications relies on several architectural principles that mixes traditional software development concepts and reviews what is new with the cloud. It is a collection of best practices, concepts, and procedures for success:

- **Design the application as a collection of (micro) services:** Cloud applications are best deployed as a collection of cloud services, or APIs. You build up from the data to the services and then combine those services into composite services or complete composite applications (either orchestration or choreography). **This is service-oriented architecture, at its essence.** Designing tightly coupled applications that focus on the user interfaces or the control, rather than exposing the underlying business functions as services will not do the job. Separate the application components physically helps on tracking and auditing the many services that make up your application. Each service (component) to be executed on the proper machine instances, service containers, service/API managers and/or governance technology. Another advantage is the service reuse from other applications or more coarse-grained services. You can break up applications into several underlying services that have value when used by other applications.

- **Decouple the data:** If you tightly couple the data to the application, it will not find a good home in the cloud. Clouds are complex distributed systems that work best with application architectures that break out processing and data into separate components. For instance, you need to reconsider the example we developed as part as Lecture 3 where a given enterprise needs to execute compute-extensive software at cloud and keeps critical data at home. However, you should be aware about the impact on the performance. Databases are reachable via Internet and that would inevitably cause latency. Specifically, database communications may determine how close your data sits to the services and applications that need to leverage it. To address that, you might consider using the huge variety of caching systems that cloud providers offer nowadays. These provide additional database performance by locally storing commonly accessed data, thereby reducing all database read requests back to the physical database.

- **Consider communications between the application components:** Chatty application components that constantly communicate with each other will lower the performance of the overall (composite) application, given that they are typically distributed over a network or the open Internet, where tolerance for high latency is desirable. You need to optimize communications between application components. For example, combine communications into a single stream of data or a group of messages, chatty application components that constantly interact with each other will lower the performance of the overall application.

- **Model, design and code for performance and scaling:** Extend considerations around how application components communicate to include overall performance as well. This includes understanding how (and when?) the application will scale up (respectively down) under an increasing (respectively decreasing) load. For instance, if 1000 or more end users log on and execute at the same time, how will (would) your application behave? Such a spike will trigger an increased traffic on the network, the increased load on the application servers, and the load placed on the back-end databases. This example might increase the load on the application servers by 80%, the load on the network by 10%, and the load on the database by 40% to 50%. Given that, adding 1000 more end users will likely saturate the application servers you have

provisioned, and, consequently, you will need to spin up more application server instances. On the other hand, the network capacity might remain the same while the number of database instances may have to increase to handle the additional load. Armed with this model, you can figure out how best to scale up/down the application by automatically spinning up/down resource instances that are needed.Of course, nearly all the cloud providers offer auto-scaling capabilities for applications and resources nowadays; however, the most efficient path lies in understanding the application's workload profile and defining the path to scaling the application, as well as, putting mechanisms in place to ensure that it will, indeed, scale. To that end, you can benefit from the several technical services provided by the cloud providers such as monitoring and logging services. For instance, monitor overall application performance and create interfaces within the application to better enable performance monitoring will help to optimize your application scalability management. However, keep in mind that the application (un)provisions resources procedures should be innate to the application rather than the cloud platform.

- **Make security systemic within the application**: When hosting an application in the cloud, **security is NOT an afterthought**. Your application architecture should make security systemic to the application. Pick a security approach and technology prior to building your application that will be effective for the type of application you are running and that will address any compliance or other data-level security issues. You might reconsider the ambulatory healthcare application presented as part as Lecture 5 where applications handle and store personal vital data. It is clear that some security rules related to the patients' identity and privacy needs to be addressed before going to the cloud. Generally speaking, cloud applications should leverage Identity and Access Management (IAM). Enterprises that develop mature IAM capabilities can reduce their security costs and, more importantly, become significantly more agile at configuring security for cloud applications. What is more, the use of IAM within cloud application provisioning will backfill into the enterprise, as these organizations modernize security approaches and technologies to align with the use of the cloud. In many cases, IAM will be provided as-a-Service to the developer/enterprise. This concept of cloud-delivered IAM quickly leads to the concept of centralized identity management. As you build more cloud applications using IAM, each application should become significantly more secure and more cost-effective. Your core objective is to design security into your application and take advantage of the native features of both the cloud and the IAM systems you use. On other hand, you need also to keep in mind that each application has its own requirements based upon the needs of the business. Security concept, requirement and even understanding often differs from one enterprise to another.

Lessons Learned (important): Building a cloud-ready application architecture requires that you pay attention to a few new things, but many of the traditional concepts are still important, such as sound design, testing, and learning from your mistakes. Most developers who provision applications on private or public cloud platforms will make some blunders, but as long as they recognize, correct, and learn from those mistakes, they will be well on their way to finding a more effective path to building applications in the cloud.

Understand that approaches such as service orientation should be given priority, even if it means longer initial application development lifecycles and bigger budgets. Even though you will probably pay more for application development in the cloud than you did for traditional

application development, the investment in services pays huge dividends year in and year out. It is a smart investment!

Task 4 (Addressing Objective 4)

1. Design and develop a simple MVC application with database access according to the tips you learned from Task 3. You need to select and motivate the choice of the storage services you selected (SQL, No-SQL, Big Data support, etc.).
2. Provision the application on a PaaS of your choice.
3. Push the code on a Git Repo and provide us with the required releases.

Task 5 (Addressing Objective 5)

1. Design and make up your own PaaS over the Proxmox infrastructure. For validation purpose, you have two alternatives:
    a. Either you provision your Resources Generator and Manager applications (developed as part as Lab 2) over that PaaS.
    b. Or, you provision the MVC application you developed as part of Task 4 of this lab.

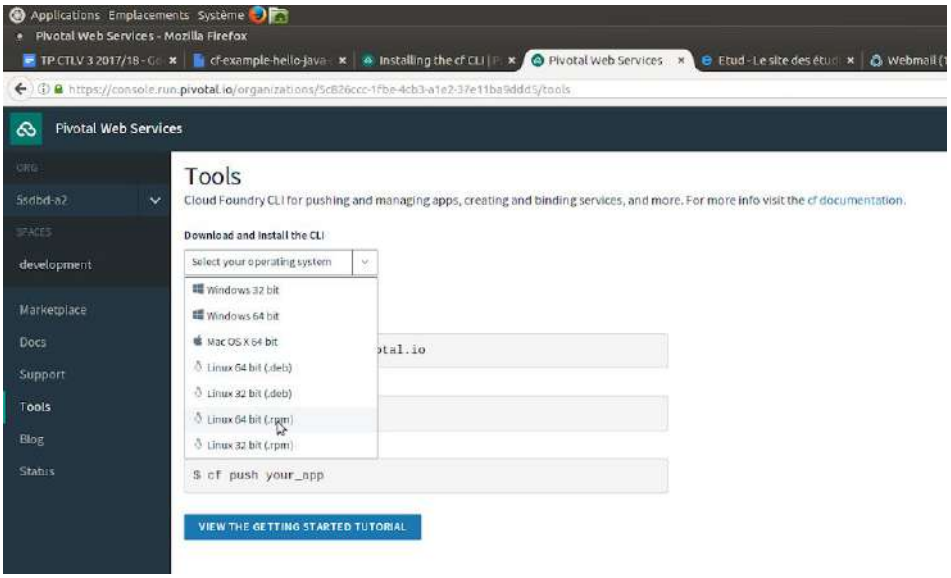
You might use a specific collection of LXC nodes to host and execute your applications while the rest as used as regular resources (to be managed). You might install and/or use remote services and/or APIs for storage.

References

1. H. Berndt, P. Graubmann, M. Wakano. “Service specification concepts in TINA-C”. In: Kugler HJ., Mullery A., Niebert N. (eds) Towards a Pan-European Telecommunication Service Infrastructure — IS&N '94. IS&N 1994. Lecture Notes in Computer Science, vol 851. Springer, Berlin, Heidelberg
2. M. Jacobs, P. Leydekkers. “Specification of Synchronization in Multimedia Conferencing Services Using the TINA Lifecycle Model,” Distrib. Sys. Eng., vol. 3, 1996, pp. 185–96.
3. S. Yangui, R.H. Glitho and C. Wette. “Approaches to end-user applications portability in the cloud: A survey”. IEEE Communications Magazine, vol. 54 No. 7, 2016, pp. 138-145.

Appendix 1

1. Download the 64bit.rpm client distribution from the CF Web console.



2. Extract the file and place your prompt under usr/bin to be able to execute cf.