Master2 iLord | Virtualisation

Openstack & Docker practical exercises

Dr Thiebolt François

TABLE OF CONTENTS

The CloudMIP platform **4**
Infrastructure 4
Openstack 4
Basic scenario | running a public instance 6
Keystone | identity service 6
Glance | image service 6
nova | launch private instance 6
neutron | public IP pool 7
ssh public key to instance 8
Horizon GUI 8
VNC connection 8

TP1 - Start instance | the CLI way **9**
0 - pre-requisites 9
tmux | a [must have] console multiplexer 10
1 - Glance, Nova and Neutron services 10
Start instance 10
map public IP to instance 11
2 - SSH connection to your instance 11
Installing a nginx server ... 12
what's my IP from abroad ? 12
3 - Horizon GUI 12

TP2 - Start instance | the API way **13**
0 - pre-requisites 13
1 - HTTP calls | check APIs 14
openstack_checks internals 14
2 - List instances | Python APIs 14
API references 14
Quick start 15
3 - Start / stop instance 15
Tips'n tricks 15
Quick start 16
Tips'n tricks continued 17

TP3 - contextualisation **18**
0 - What's all this cloud-init stuff, Anyhow? 18
1 - Contextualization | the CLI way 19
user-data script 19
start customized instance 19
customization | inner view 20

TP4 - Docker **21**

0 - What's all this containers stuff, Anyhow? 21
docker images 21
FS mounts and network ports bindings 22
docker cli 22
1 - Docker setup @ instance 23
the VXLAN issue! 23
docker-ce (Community Edition) setup 23
2 - Basic docker commands 24
New set of docker commands 26
Network in dockerized environment 26
Saving a customized container 26
3 - docker volumes 27

TP5 - Docker swarm / Kubernetes 28
0 - Principles 28
1 - Start a docker swarm manager 28
2 - Add docker hosts to the swarm 29
3 - Launch a simple app 30
docker service create 30
4 - Fedora Atomic 31
5 - Bonus 31

Annexe-1 Openstack addon commands 32
create image from instance | nova snapshot 32
Add Fedora atomic image 32
Persistent storage | cinder volumes 32

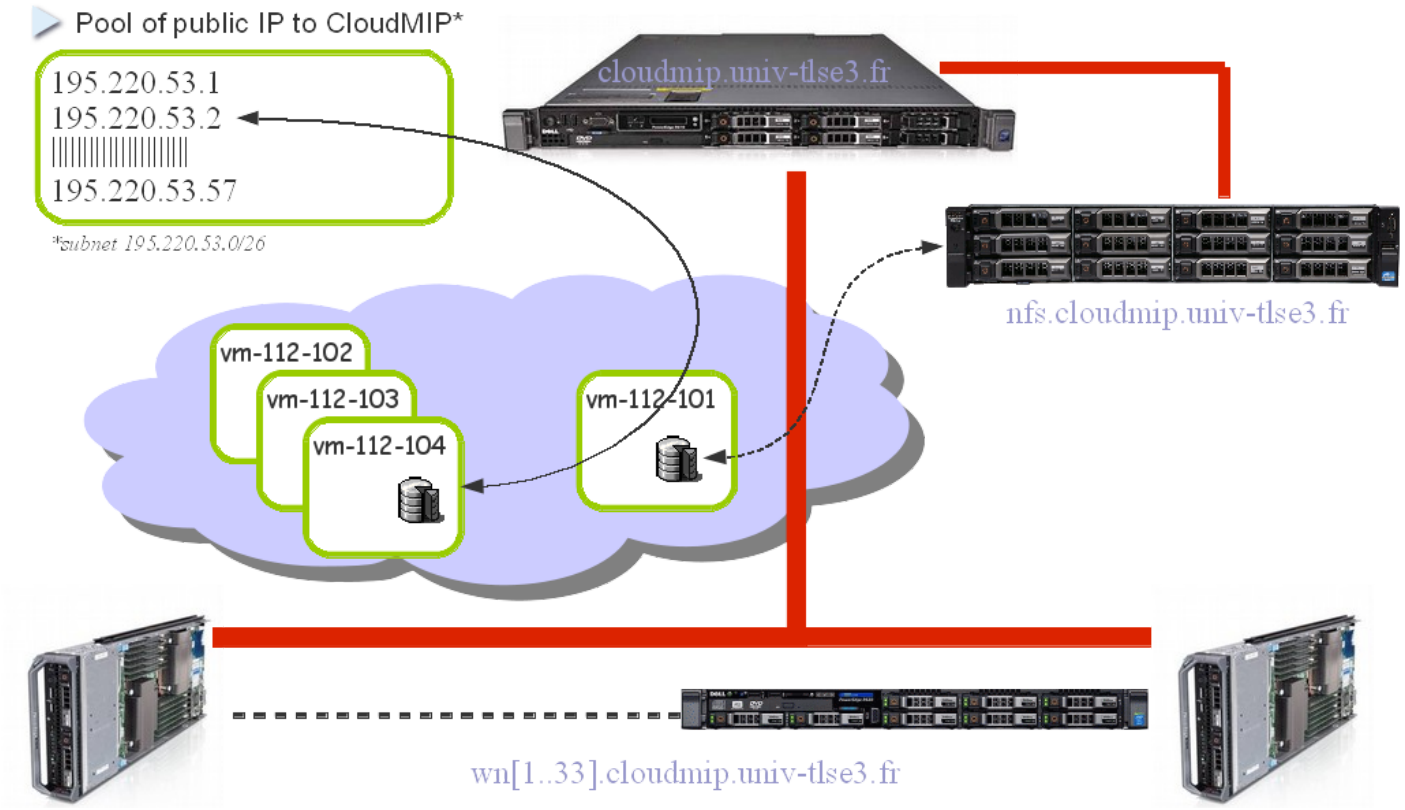
Annexe-2 Docker commands 34
[DEPRECATED] docker config files 35

The CloudMIP platform

All along these practical exercises, we'll make use of the France-Grilles CloudMIP platform. France-Grille is a federation of cloud platforms that spread across more than 10 sites in France and Luxembourg. CloudMIP is a 280 cores, 2TB ram and 45TB storage platform hosted at the datacenter of our university.

Infrastructure

Below is an infrastructure overview of the whole platform.



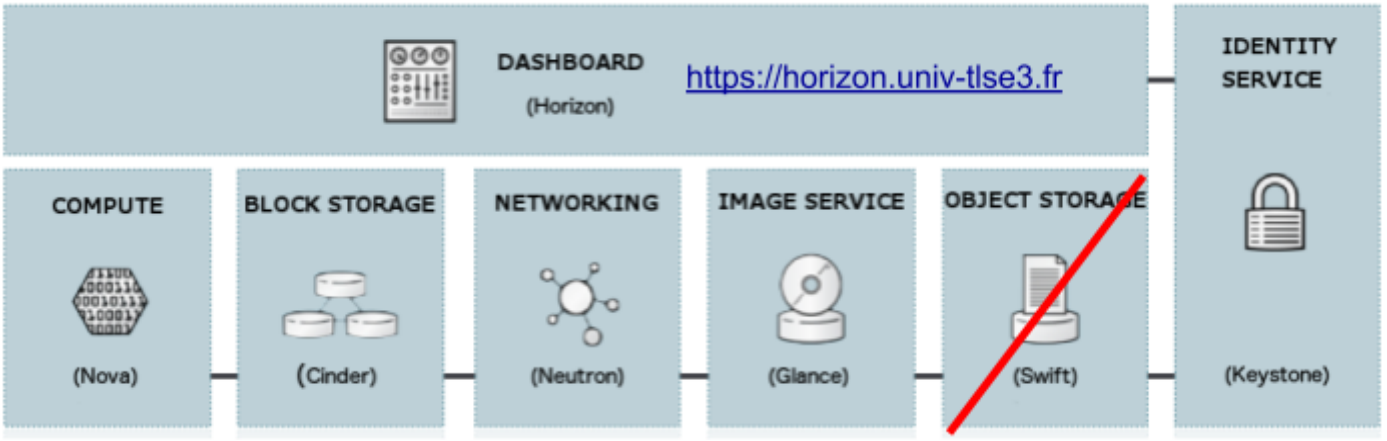
Please have a look to our Wiki for a more detailed hardware description along with others stuff related to your practical exercises at https://cloudmip.univ-tlse3.fr/platform/platform

Openstack

Being a production-grade cloud platform, CloudMIP has been installed with OpenStack Liberty (2015). I setup some of the essentials services to allow users to start instances (i.e virtual machines) and to enable them to save data in some persistent storage (the Cinder service).

Take note that Openstack is community-driven software. On the positive side, there's a fast evolution of components (eg. Keystone, Neutron ...) with drawbacks like inconsistent documentation and obscure configuration files :|

We present below some of the common services found in almost all openstack platforms.



Service	Point d'accès au service
Compute	http://frontal:8774/v2/dfc44d41d9384a6caa0963ffba713fad
Network	http://frontal:9696
Volumev2	http://frontal:8776/v2/dfc44d41d9384a6caa0963ffba713fad
Image	http://frontal:9292
Volume	http://frontal:8776/v1/dfc44d41d9384a6caa0963ffba713fad
Identity	http://frontal:5000/v2.0
Affichage de 6 éléments	

Each service is reachable from GUI (https://horizon.univ-tlse3.fr), CLI and API.

Openstack services end-points API

Service	API ENV. VARIABLE	Openstack service name
Compute	$OS_COMPUTE_API	nova
Network	$OS_NETWORK_API	neutron
Volume / VolumeV2	$OS_STORAGE_API	cinder
Image	$OS_IMAGE_API	glance
Identity	$OS_AUTH_URL	keystone

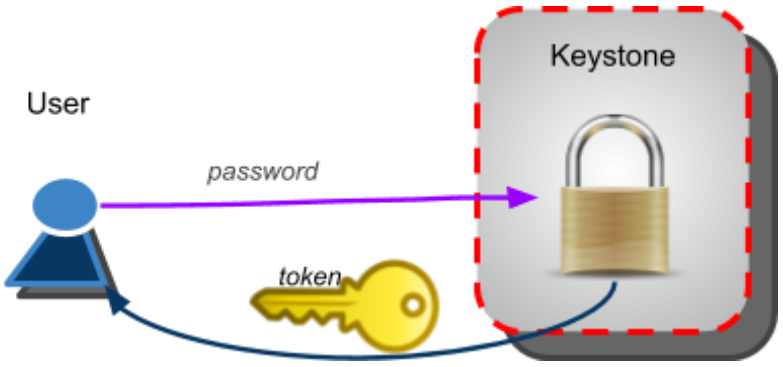
Basic scenario | running a public instance

The most common use case is launching an instance that will be reachable from the Internet. However, before undertaking any action on the CloudMIP platform, you must first obtain an account ;)

Keystone | identity service

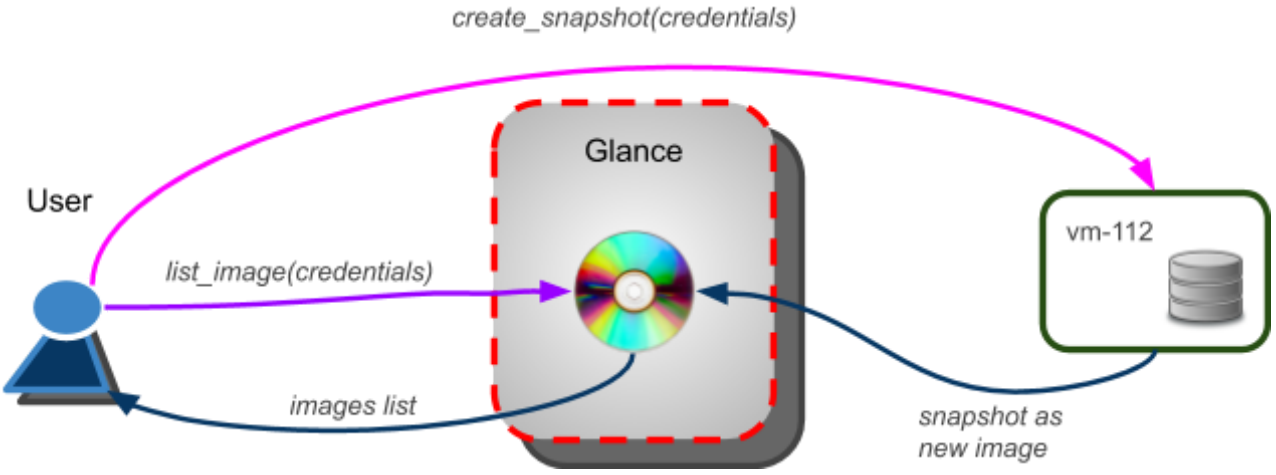
You need credentials to interact with any openstack service. These credentials may be a password or a time-limited token.

Whatever command you enter, credentials are needed!



Glance | image service

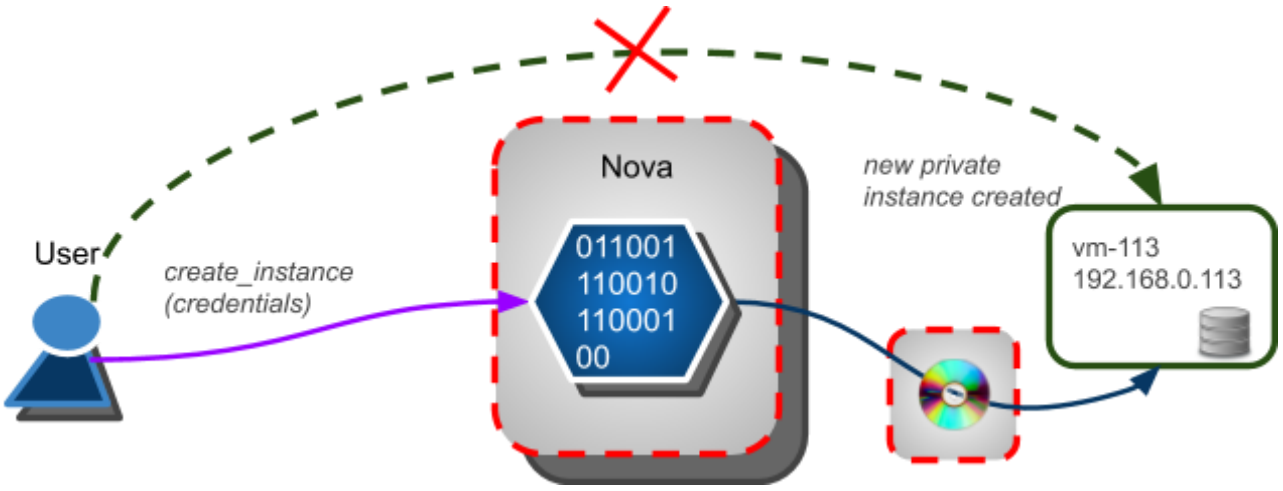
Before launching an instance, an image to instantiate ought to exist.



An image can be downloaded and integrated within the glance service. On the other side, a running instance may be snapshotted thus generating a new image that could be instantiated afterward.

nova | launch private instance

Having registered a public key, a glance image has been selected and we launch an instance featuring a private network IP (192.168.0.x/24)



However, it is impossible for a regular user to gain access to its private VM because CloudMIP does not implements a flat network but makes use of VXLAN that comes with Linux network namespaces.

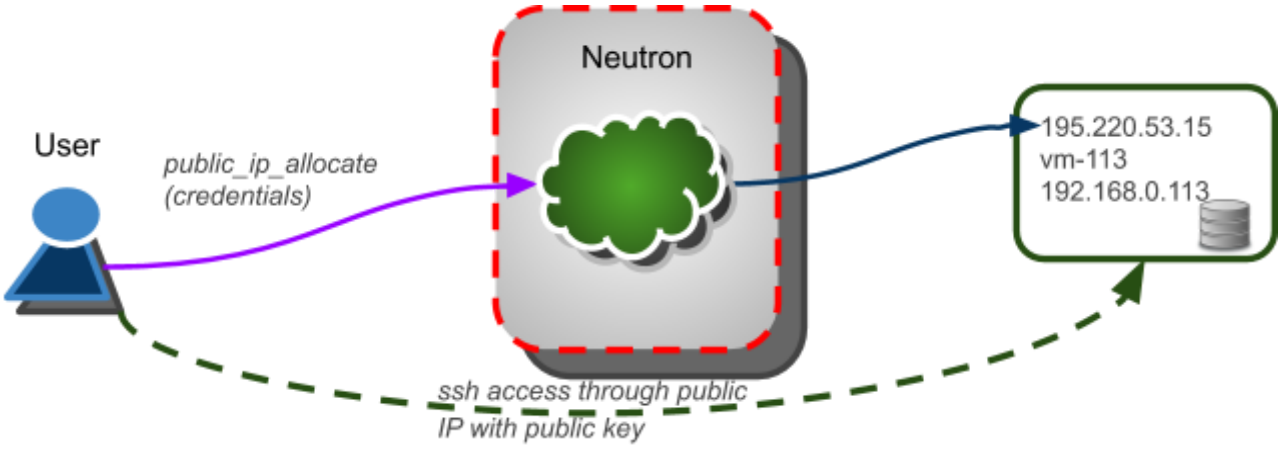
When using a flat network (usually combined with NAT at front-node), instances only feature a private IP and it will be almost impossible to track users access in case we're requested by authorities :(

neutron | public IP pool

Openstack network is managed by the Neutron service. This service holds both the **'private'** and **'public'** networks at CloudMIP.

Network	IP span
private	192.168.0.0/24
public	195.220.53.1 → 195.220.53.40

Thus we need to allocate a public IP and then we'll map it on our instance.



ssh public key to instance

Default images does not allow password access to virtual machines … thus you need to make use of the private key associated with the public key you registered with openstack.

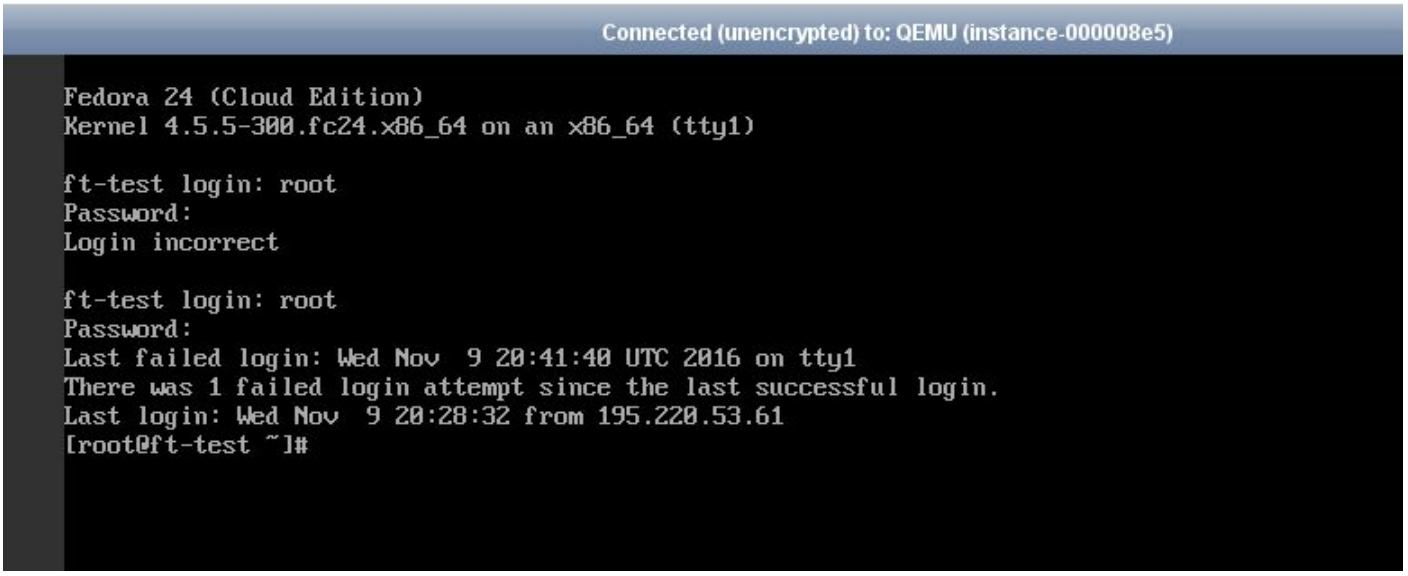
Horizon GUI

To see what's happening with your instance or to see things at the platform level:

https://horizon.univ-tlse3.fr

VNC connection

Please note that you can also connect to your instance through VNC:
Project → Compute → Instance → Action (from your instance row) → Console



Note: QWERTY keyboard!

>>> don't forget to terminate instances (i.e nova delete) when you're done <<<

TP1 - Start instance | the CLI way

Openstack general principles have been explained, so you'll now exercises them on the CloudMIP platform through the command line interface (CLI).

0 - pre-requisites

In order to ease interactions with the various openstack services, there's some setup required.

First, connect to the platform,

```
ssh <user>@cloudmip.univ-tlse3.fr
```

… then change passwd right now

```
passwd
```

then execute the command that will prepare start files,

```
osSetCredential.sh
```

WARNING: this command will create an environment variable named OS_PASSWORD. This variable will hold your **cleartext** password in your ~/.env file → ensure that **no one else** can access it!

[ONLY FIRST TIME] … import env. variables in current shell

```
source ~/.env
```

We'll now build a (public, private) keys tandem ...

```
ssh-keygen -t rsa
```

Note: just press 'enter' to any question!

```
nova keypair-list
nova keypair-delete mykey
```

*Note: you may **delete ALL existing keys** (last year students!)*

and finally add **your** generated public key to your nova profile:

```
nova keypair-add --pub-key ~/.ssh/id_rsa.pub mykey
```

Now you can check that your SSH key has been integrated:

```
nova keypair-list
```

tmux | a [must have] console multiplexer

- console multiplexer → `tmux`

This tool enables you to launch console with horizontal, vertical splits, multi window … everything in a single terminal
TMUX basic commands (CTRL A for command mode) CTRL a + " → horizontal split CTRL a + % → vertical split CTRL a + CTRL a → switch pane CTRL a + d → detach (apps continue to run) CTRL a +) → next session CTRL a + c → create new window in current session CTRL a + n → next window from current session CTRL a + p → previous window from current session
tmux ls → list tmux sessions tmux att → attach to first session

Note: `.tmux.conf` file in your home redefines `tmux` to match `screen` behaviour

1 - Glance, Nova and Neutron services

We'll now launch an instance, the same way we've done through the GUI.

List available images

```
glance image-list
```

List running instance

```
nova list
```

List networks

```
neutron net-list
```

… and pay attention to the private network ID

Start instance

Before launching an instance, we ought to know some key parameters like flavour, image and network to use:

```
nova flavor-list
nova image-list
neutron net-list
```

Note that **only private network** is reachable from compute nodes so this is the one your instances ever ought to use at startup:

```
nova boot --flavor m1.small --image COS75 --nic net-id=<NETWORK_ID> \
--security-group default --key-name mykey instance_name
```

```
nova boot --flavor m1.small --image COS75 --nic
net-id=c1445469-4640-4c5a-ad86-9c0cb6650cca --security-group default --key-name
mykey COS75_${OS_USERNAME}
```

```
nova list
```

… all instance from OS_PROJECT_NAME will appear

after a while, your VM ought to get started, you can then ask for additional details

```
nova show <instance_ID | instance_name>
```

… and you can also ask for 'console logs'

```
nova console-log <instance_ID | instance_name>
```

map public IP to instance

To enable instance to get reached from the internet, we grab a floating-ip
But before going-on with the floating public IP creation, is there any already created public IP available ?

```
nova floating-ip-list
```

*Ok, **if there isn't** a Public IP available, let's create one*

```
neutron floatingip-create public
Created a new floatingip:
+---------------------+--------------------------------------+
| Field               | Value                                |
+---------------------+--------------------------------------+
| fixed_ip_address    |                                      |
| floating_ip_address | 195.220.53.4                         |
| floating_network_id | c254d472-6cfd-425a-9960-e9d38ea4c391 |
| id                  | b7015888-9dde-4273-a377-631fd4f235ac |
| port_id             |                                      |
| router_id           |                                      |
| status              | DOWN                                 |
| tenant_id           | 6ac3b0c5fd5641928a412ed2b0ad65e5     |
+---------------------+--------------------------------------+
```

```
nova floating-ip-associate <instance_ID | instance_name> <allocated public IP>
```

2 - SSH connection to your instance

Start a shell. Then, by means of your private key (the one associated to your public key that has been pre-loaded within your instance), you will get connected to your instance:

```
ssh -i <path to your private key> centos@<public IP>
```

```
ssh centos@<allocated public IP>
```

... well done player one ;)
Note: the `'centos'` username is the default one set by packagers of the CentOS openstack image.

then to gain root access

```
[~]$ sudo su -
[~]#
```

Installing a nginx server ...

you can now install some additional software

```
yum -y install nginx
systemctl enable nginx
systemctl start nginx
```

now you can check with http://<your_public_IP>

what's my IP from abroad ?

```
yum -y install bind-utils
```

```
dig +short myip.opendns.com @resolver1.opendns.com
```

```
curl ipinfo.io/ip
```

3 - Horizon GUI

Check resources online

https://horizon.univ-tlse3.fr

>>> don't forget to terminate instances (i.e nova delete) when you're done! <<<

TP2 - Start instance | the API way

We'll now launch instances using the Openstack APIs.
As a first step, you ought to retrieve the various Openstack API services end-points.
Log with your account to https://horizon.univ-tlse3.fr

Projet → Compute → Accès et sécurité → onglet Accès API

0 - pre-requisites

Most python scripts we'll develop in these practical exercises will need some essential informations like **authentication tokens** for example.

[Quick start] download '`openstack.sh`' file from

https://cloudmip.univ-tlse3.fr Openstack → download_area

or *`wget https://cloudmip.univ-tlse3.fr/teaching/francois/openstack.sh`*

… then execute this script to set variables in your shell

```
source ~/openstack.sh
```

Now, we'll need a **token** that enables us to avoid sending username / password for every command

```
xxx@frontal[~] openstack token issue
password:
+------------+----------------------------------+
| Field      | Value                            |
+------------+----------------------------------+
| expires    | xxxx-xx-xxT20:04:58Z             |
| id         | fbc84bccd8f74e60af059ef83a5e7220 |
| project_id | dfc44d41d9384a6caa0963ffba713fad |
| user_id    | xxx                              |
+------------+----------------------------------+
```

[alternate scriptable solution]

```
export OS_TOKEN=$(openstack token issue -f value -c id)
export OS_URL="${OS_AUTH_URL}"
```

*Note: when using **tokens**, you must provide OS_URL env. var. for CLI commands*

1 - HTTP calls | check APIs

All Openstack APIs are reachable through HTTP GET / POST requests.

[Quick start] download '`openstack_checks.sh`' file from

https://cloudmip.univ-tlse3.fr Openstack → download_area

or *`wget https://cloudmip.univ-tlse3.fr/teaching/francois/openstack_checks.sh`*

… then look at commands within

```
cat ~/openstack_checks.sh
source ~/openstack_checks.sh
```

openstack_checks internals

To list images available for launching instances

```
curl -s -H "X-Auth-Token: $OS_TOKEN" ${OS_COMPUTE_API}/images | python -m json.tool
```

Note: same result is achieved with cli '`glance image-list`'

Details of all of the available flavors

```
curl -s -H "X-Auth-Token: $OS_TOKEN" $OS_COMPUTE_API/flavors/detail | python -m json.tool
```

List of running instances

```
curl -s -H "X-Auth-Token: $OS_TOKEN" $OS_COMPUTE_API/servers | python -m json.tool
```

2 - List instances | Python APIs

As a first step, you'll have to implement a simple python application that will:

- list running instances (i.e servers) along with all of their IPs.

To achieve this, we'll make use of the Nova python client and previously defined environment variables
http://developer.openstack.org/api-ref/compute/

API references

Links to OpenStack APIs documentations
http://developer.openstack.org/api-guide/quick-start/api-quick-start.html

http://docs.openstack.org/user-guide/sdk.html
http://docs.openstack.org/developer/python-keystoneclient/
http://docs.openstack.org/developer/python-keystoneclient/using-api-v2.html
http://docs.openstack.org/developer/python-novaclient

Quick start

Download '`tp2_listInstances.py`' file from

https://cloudmip.univ-tlse3.fr Openstack → download_area

or `wget https://cloudmip.univ-tlse3.fr/teaching/francois/python/tp2_listInstances.py`

… then you need to complete the file …

… and once you're done, test it!

```
chmod a+x ./tp2_listInstances.py
./tp2_listInstances.py
```

3 - Start / stop instance

Next, you'll start to write a new python application `tp2_startStopInstance.py` that would have to:

- start a private instance (i.e no public IP, only a private one),
- list instances,
- destroy our private instance.

However, before starting an instance, you need to select:

- flavour to use (i.e size of instance ---CPU,RAM etc),
- image to instantiate (see command `glance image-list`),
- which network to get tied to (see command `neutron net-list`).

Tips'n tricks

Openstack documentation about Python API may appear terse or misleading, that's why you always ought to check about the currently installed libraries.

- python `__file__` attribute

```
francois@frontal[~] python
Python 2.7.5 (default, Aug 18 2016, 15:58:25)
[GCC 4.8.5 20150623 (Red Hat 4.8.5-4)] on linux2
Type "help", "copyright", "credits" or "license" for more information.
>>> import novaclient
>>> novaclient.__file__
'/usr/lib/python2.7/site-packages/novaclient/__init__.pyc'
>>>
```

- thus, search for `'create'` and `'start'` methods from novaclient API file

```
/usr/lib/python2.7/site-packages/novaclient/v2/server.py
```

You may also have a look to the test units of the novaclient itself:

```
/usr/lib/python2.7/site-packages/novaclient/tests/unit/v2/test_servers.py
```

- python built-in `dir(<object>)` method → return objects attributes

```
francois@frontal[~] python
Python 2.7.5 (default, Aug 18 2016, 15:58:25)
[GCC 4.8.5 20150623 (Red Hat 4.8.5-4)] on linux2
Type "help", "copyright", "credits" or "license" for more information.
>>> from novaclient.client import Client as computeClient
>>> compute=computeClient(2)
>>> dir(compute)
['__class__', '__delattr__', '__dict__', '__doc__', '__enter__', '__exit__',
'__format__', '__getattribute__', '__hash__', '__init__', '__module__',
'__new__', '__reduce__', '__reduce_ex__', '__repr__', '__setattr__',
'__sizeof__', '__st
r__', '__subclasshook__', '__weakref__', 'agents', 'aggregates',
'api_version', 'authenticate', 'availability_zones', 'certs', 'client',
'cloudpipe', 'dns_domains', 'dns_entries', 'fixed_ips', 'flavor_access',
'flavors', 'floating_ip_pools', 'floating_ips', 'floating_ips_bulk',
'fping', 'get_timings', 'hosts', 'hypervisor_stats', 'hypervisors',
'images', 'keypairs', 'limits', 'networks', 'os_cache', 'projectid',
'quota_classes', 'quotas', 'reset_timings', 'security_group_default_rules',
'security_group_rules', 'security_groups', 'server_groups', 'servers',
'services', 'set_management_url', 'tenant_id', 'usage', 'user_id',
'versions', 'virtual_interfaces', 'volume_snapshots', 'volume_types',
'volumes']
>>> dir(compute.flavors)
['__abstractmethods__', '__class__', '__delattr__', '__dict__', '__doc__',
'__format__', '__getattribute__', '__hash__', '__init__', '__module__',
'__new__', '__reduce__', '__reduce_ex__', '__repr__', '__setattr__',
'__sizeof__', '__str__', '__subclasshook__', '__weakref__', '_abc_cache',
'_abc_negative_cache', '_abc_negative_cache_version', '_abc_registry',
'_build_body', '_create', '_delete', '_get', '_hooks_map', '_list',
'_update', 'add_hook', 'alternate_service_type', 'api', 'api_version',
'cache_lock', 'client', 'completion_cache', 'create', 'delete', 'find',
'findall', 'get', 'is_alphanum_id_allowed', 'list', 'resource_class',
'run_hooks', 'write_to_completion_cache']
```

Quick start

Download '`tp2_startStopInstances.py`' file from

https://cloudmip.univ-tlse3.fr Openstack → download_area

or `wget https://cloudmip.univ-tlse3.fr/teaching/francois/python/tp2_startStopInstance.py`

… you then need to complete the file … (have a look to the following link)
https://docs.openstack.org/python-novaclient/latest/reference/api/novaclient.v2.servers.html

… and once you're done, test it!

```
chmod a+x ./tp2_startStopInstances.py
./tp2_startStopInstances.py
```

Tips'n tricks continued

Once you'll have made a call to the `'create'` method (starts an instance), the object returned will exhibits the following attributes:

```
['HUMAN_ID', 'NAME_ATTR', 'OS-DCF:diskConfig', '__class__', '__delattr__',
'__dict__', '__doc__', '__eq__', '__format__', '__getattr__',
'__getattribute__', '__hash__', '__init__', '__module__', '__new__',
'__reduce__', '__reduce_ex__', '__repr__', '__setattr__', '__sizeof__',
'__str__', '__subclasshook__', '__weakref__', '_add_details', '_info',
'_loaded', 'add_fixed_ip', 'add_floating_ip', 'add_security_group',
'adminPass', 'backup',  'change_password', 'clear_password',
'confirm_resize', 'create_image', 'delete', 'diagnostics', 'evacuate',
'force_delete', 'get', 'get_console_output', 'get_password',
'get_rdp_console', 'get_serial_console', 'get_spice_console',
'get_vnc_console', 'human_id', 'id', 'interface_attach', 'interface_detach',
'interface_list', 'is_loaded', 'links',
'list_security_group', 'live_migrate', 'lock', 'manager', 'migrate',
'networks', 'pause', 'reboot', 'rebuild', 'remove_fixed_ip',
'remove_floating_ip', 'remove_security_group', 'rescue', 'reset_network',
'reset_state', 'resize', 'restore', 'resume', 'revert_resize',
'security_groups', 'set_loaded', 'shelve', 'shelve_offload', 'start',
'stop', 'suspend', 'to_dict', 'unlock', 'unpause', 'unrescue', 'unshelve',
'update']
```

>>> don't forget to terminate instances (i.e nova delete) when you're done! <<<

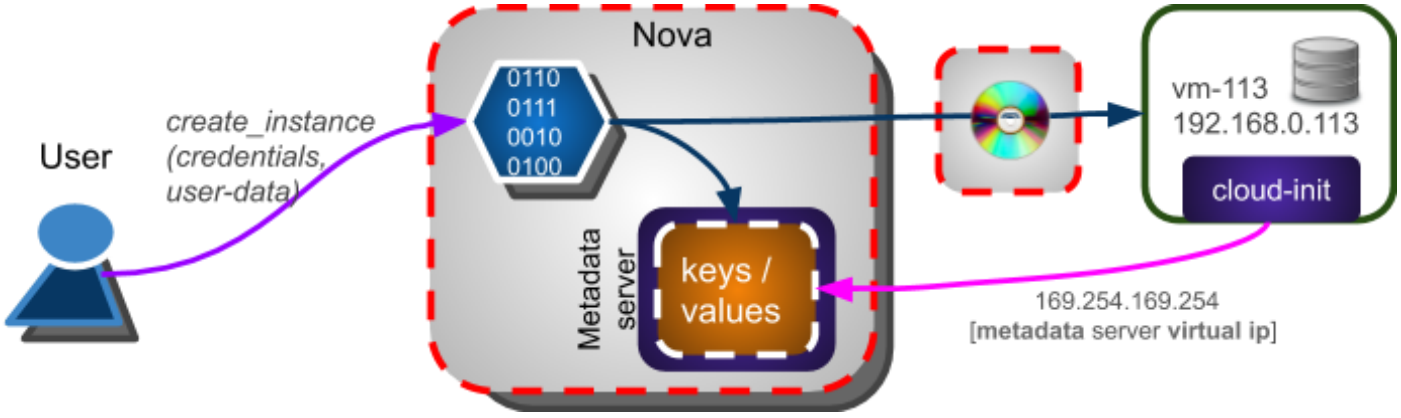
TP3 - contextualisation

Whenever you start an instance, its is based on an image that features some pre-installed software along with some predefined services to start at boot. Contextualisation is the process that leads to instance customization: even based on the same image, instances may exhibit different behaviour according to the configuration sent through **cloud-init**.
http://cloudinit.readthedocs.io/en/latest/topics/datasources/openstack.html

0 - What's all this cloud-init stuff, Anyhow?

Openstack images usually contains **cloud-init** software that enables instances customisation. End-user activate such customization through `'user_data'` option that gets stored within the metadata service (proxied to nova service). Thus, cloud-init software that runs on instance will get in touch with the openstack metadata service at **169.254.169.254** [metadata server virtual ip] to retrieve its configuration along with others metadata like public_IP, hostname, ssh_keys ...

https://raymii.org/s/tutorials/Automating_Openstack_with_Cloud_init_run_a_script_on_VMs_first_boot.html



Below is a summary of the various ways to instances customization:

- `nova help boot`

```
--meta <key=value>              Record arbitrary key/value metadata to
                                /meta_data.json on the metadata server. Can be
                                specified multiple times.
--file <dst-path=src-path>      Store arbitrary files from <src-path> locally
                                to <dst-path> on the new server. Limited by
                                the injected_files quota value.
--user-data <user-data>         user data file to pass to be exposed by the
                                metadata server.
```

While cloud-init is a rather complex software, in earlier times, we used to pack everything an instance was in need of, to a virtual CDrom. This cdrom was subsequently mounted (read-only) as an additional device (e.g /dev/vdb) and read by contextualization scripts.

1 - Contextualization | the CLI way

We'd like you to customize a COS75 based instance in a way that it will exhibit the following:

- a nginx server (i.e http server)

As for TP1-1 Start instance, you will instantiate a small VM, allocate a public IP, create and pass a customization script (i.e `--user-data` file) and access to its web server.

user-data script

You will write a shell script on your own that will grab all the commands issued in TP1 when you were connected to your VM:

- `tp3_user-data.sh`

```
#!/bin/bash
yum -y instal epel-release
yum -y install nginx
systemctl start nginx
```

```
chmod a+x tp3_user-data.sh
```

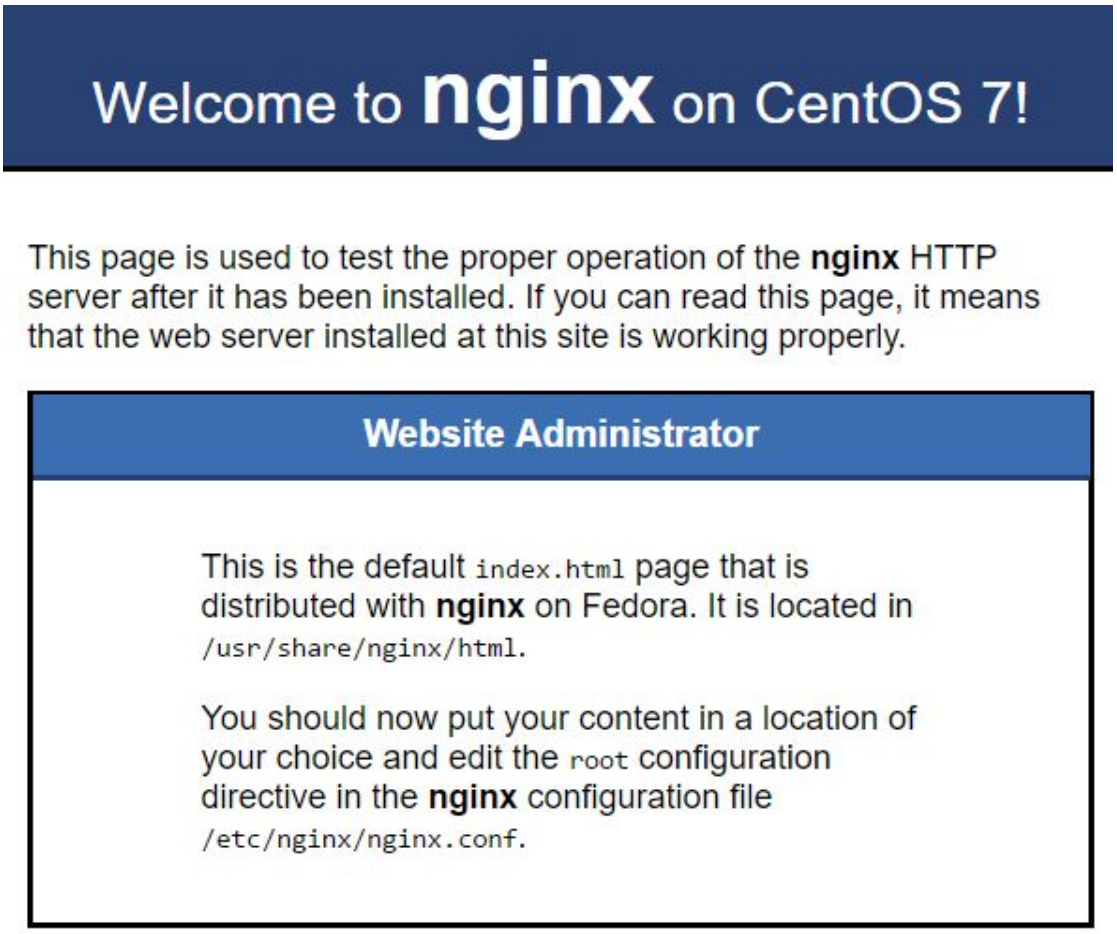
start customized instance

Launch instance with `--user_data` field and a ***custom*** metadata through `--meta` option

```
nova boot --flavor m1.small --image COS75 \
--nic net-id=c1445469-4640-4c5a-ad86-9c0cb6650cca --security-group default \
--meta answer=42 \
--key-name mykey \
--user-data tp3_user-data.sh \
COS75_${OS_USERNAME}
```

... allocate and map a public IP to your private instance … (see map public IP to instance)

… then check access to its HTTP server via a browser (or curl, wget …)



customization | inner view

Ok, now that you get SSH connected to your VM, type the following command:

```
#> curl http://169.254.169.254/openstack/latest
meta_data.json
user_data
password
vendor_data.json
network_data.json
```

- retrieve `user_data` script sent at instantiation

```
curl http://169.254.169.254/openstack/latest/user_data -O
cat user_data
```

- to ease watching JSON data structures ...

```
yum -y install python
```

- exemple: watch `meta_data` structure

```
curl http://169.254.169.254/openstack/latest/meta_data.json | python -m json.tool
```

>>> don't forget to terminate instances (i.e nova delete) when you're done! <<<

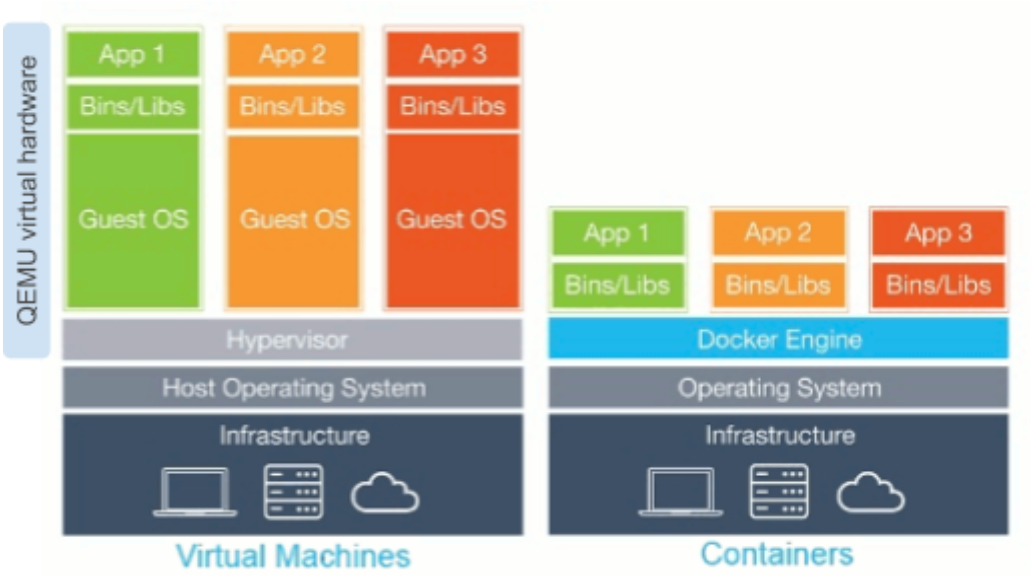
TP4 - Docker

We'll now start to learn about what had started to be a revolution within clouds: **containers**!
A container is a kind of ultra lightweight instance featuring:

- milliseconds to boot,
- one application to run (without deployment hassle),
- better resources utilization (i.e than instances),
- (almost) no overhead.

0 - What's all this containers stuff, Anyhow?

In many use cases, users apply for (large) instances (i.e high CPU count and memory) … to finally just launch a single application … thus wasting dedicated resources :(



Container's secret: linux **namespaces**!

Instead of a complete isolation through an emulated hardware along with a dedicated OS … let's run all containers applications on the same host but in different **namespaces** (ip netns, cgroups …) … that's thin insulation!

docker images

Before starting an application within a container, you first ought to retrieve an image.

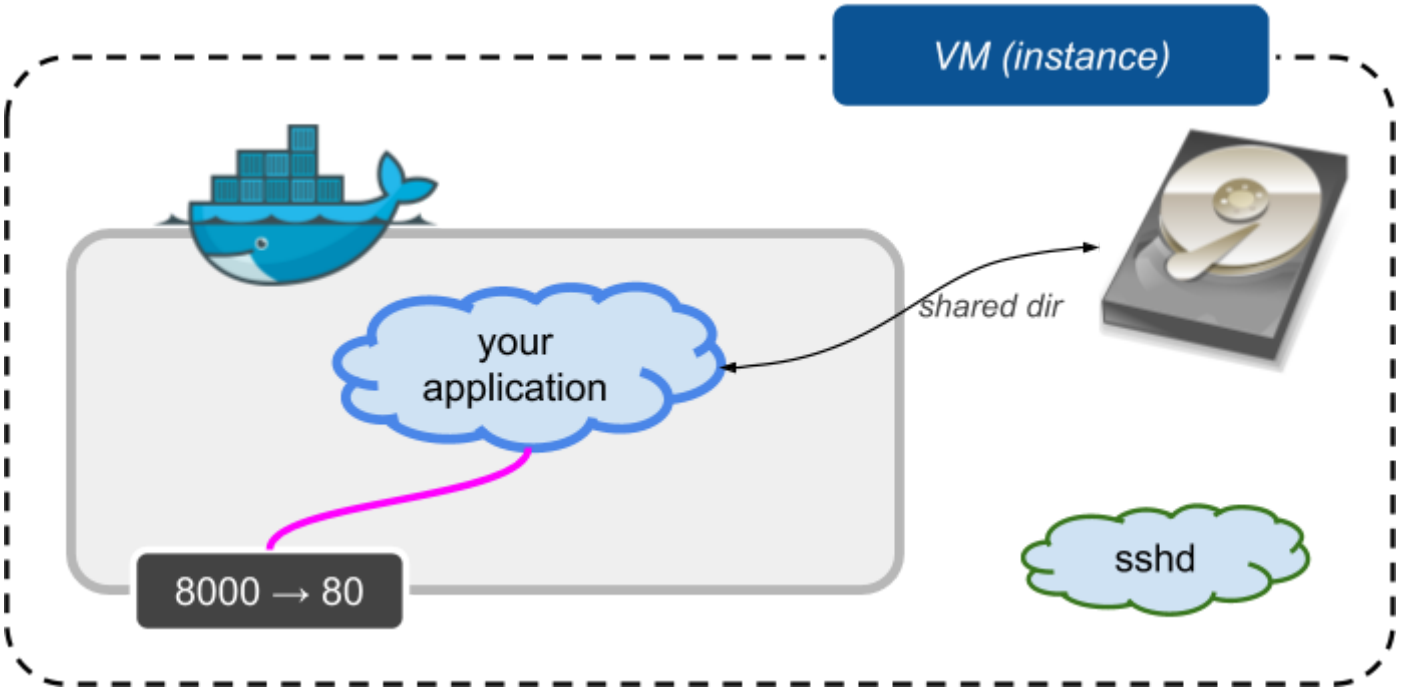
```
docker pull fedora
```

… coming from https://hub.docker.com (no need for any account)

Your host's OS doesn't matter: here we decided that our next application will get run with Fedora libraries! That's because containerized applications & libraries only make use of kernel syscalls from host.

FS mounts and network ports bindings

In the next picture, we present a traditional use case: exposing an application isolated in a container via a network port reachable from its host (here an instance). The application in the container runs a service on port 8000. This port is mapped to port 80 on host (instance) … thus a http access to this instance will get forwarded to port 8000 in container.



Behind the hood is a bridge `docker0` to whom all containers get attached to with private IPs (172.17.0.0**/16**). In addition, iptables rules map ports to/from host.

docker cli

In a previous example, we decided to mount a directory from the host to our container (possibly `/etc/shadow`)… that's one of the reasons why `docker` is a **root-only** command.
Later, we'll discover how to give users access to a container.

In addition to what we'll be talking later, you may start to have a look at this excellent docker introduction from Sébastien Binet:
http://m2siame.univ-tlse3.fr/teaching/francois/docker-introduction_dec15.pdf

1 - Docker setup @ instance

Before going on with docker setup in your instance … you first ought to **start an instance** featuring a public IP (same as seen earlier) …

the VXLAN issue!

CloudMIP (like many others cloud platforms) make use of Virtual eXtensible LAN (i.e VXLANs). A vxlan enables overlay networks leading to up to 16 million virtual vlan in a physical vlan. It **encapsulates** layer2 Ethernet frames within layer 4 UDP frames. VXLAN endpoints, which terminate VXLAN tunnels are known as VXLAN tunnel endpoints (VTEPs) and make use of port 4789.
https://en.wikipedia.org/wiki/Virtual_Extensible_LAN

```
[root@cos75-francois ~]# yum -y install net-tools bind-utils
[root@cos75-francois ~]# ifconfig
eth0: flags=4163<UP,BROADCAST,RUNNING,MULTICAST>  mtu 1450
      inet 192.168.0.167  netmask 255.255.255.0  broadcast 192.168.0.255
      inet6 fe80::f816:3eff:fe15:6276  prefixlen 64  scopeid 0x20<link>
      ether fa:16:3e:15:62:76  txqueuelen 1000  (Ethernet)
      ……
```

… thus, since there's encapsulation, it means that MTU is shaped down from 1500 to **1450** ...

docker-ce (Community Edition) setup

install software ...

```
yum -y install yum-utils
yum-config-manager --add-repo https://download.docker.com/linux/centos/docker-ce.repo
yum -y install docker-ce
```

create a directory for dockyard (instead of /var/lib/docker) and set it as default docker dir

```
mkdir /dockyard
```

create docker daemon config file

```
mkdir /etc/docker
```

- `/etc/docker/daemon.json`

```
{
    "log-driver": "journald",
    "data-root": "/dockyard",
    "storage-driver": "overlay2",
    "storage-opts": [
        "overlay2.override_kernel_check=true"
    ],
    "bip": "172.17.0.1/24",
    "mtu": 1450,
    "fixed-cidr": "172.17.0.0/24"
}
```

***Note**: BEWARE of **invisible** chars when you copy-paste from PDF files!*

… then enable and start service

```
systemctl enable docker
systemctl start docker
```

… watch journal

```
journalctl -efu docker
```

… and test it is functional

```
docker info
docker run hello-world
```

2 - Basic docker commands

Find proper image …

```
docker search fedora
```

Note: Alpine Linux is the default Linux system installed in almost all constrained systems

retrieve latest fedora container (or specific version)

```
docker pull fedora
or docker pull fedora:latest
or docker pull fedora:<release>
```

★ What happened ? Why do we need such docker pull command ?

```
docker images
REPOSITORY          TAG                 IMAGE ID            CREATED             VIRTUAL SIZE
docker.io/fedora    latest              3fc68076e184        5 weeks ago         206.3 MB
```

★ From a filesystem point-of-view, where's located such fedora image ?

start **tmux** and launch a shell in our newly grabbed container image in an interactive mode!

```
docker run --rm -it fedora /bin/bash
```

★ Where have you been teleported ? what the '--rm' option means ?

… within container …

```
#/ dnf -y install python3-pip
#/ pip3 search django
#/ sleep 600
```

detach from container (i.e get back to tmux)

```
CTRL  p   CTRL  q
```

★ so you just installed some software within your container, what about the size of the image this container has been started from ?

```
ps -elf | grep -i sleep
```

★ What can you see ? Could you explain what's happening ??

It's now time for you to test some commands

```
docker ps
docker ps -a
docker ps -l
docker ps -ql
```

★ What can you conclude about the states of the various containers ?

Now re-attach to your fedora container and execute **`exit`** *from container shell prompt*

```
docker attach $(docker ps -ql)
#/ exit
```

New set of docker commands

```
docker run -d fedora sleep 600
docker exec -it $(docker ps -ql) bash
#/ echo "fastestmirror=True" >> /etc/dnf/dnf.conf
#/ dnf -y install procps
#/ ps -elf
```

★ What did we do here ? what are these processes we can see in response to the latest command ?

Network in dockerized environment

```
docker run -d fedora sleep 500
docker inspect $(docker ps -ql)
```

★ How to ping your running docker ? what can you say about the kernel's routing table ?

```
brctl show
arp
```

★ explain results obtained against each of the commands' outputs

Saving a customized container

```
docker run -it fedora /bin/bash
```

```
#/ echo "fastestmirror=True" >> /etc/dnf/dnf.conf
#/ dnf -y --allowerasing update
#/ exit
```

```
docker commit -a $(whoami) -m "fedora_updated" $(docker ps -ql) test/fedora:myversion
```

```
docker images
docker history fedora:myversion
```

★ what do you conclude about our container ? about the generated image ? how could you use this new image ?

okay, it's now time to stop and destroy all containers and to delete all images

```
docker stop <id>
docker ps -a
docker rm <id>
docker rmi <imageID>
```

… and a lot of useful Docker commands

3 - docker volumes

A volume is a persistent storage that enables data survival across containers lifecycle.

```
docker volume create -d local --name pgsql -o size=20G
```

```
docker volume ls
docker volume inspect pgsql
```

★ From a filesystem point of view, where is this volume ?

It's now up to you to link this volume into a new container and to save data within

```
???
```

Hint: `docker -v <volume_name>:<target_dir> …`

… and finally destroy your container along with this volume

```
docker volume rm pgsql
```

>>> at this step, you have a functional docker node (clean a bit things) we'll use hereafter <<<
>>> keep your connection to this instance in a tmux session! <<<

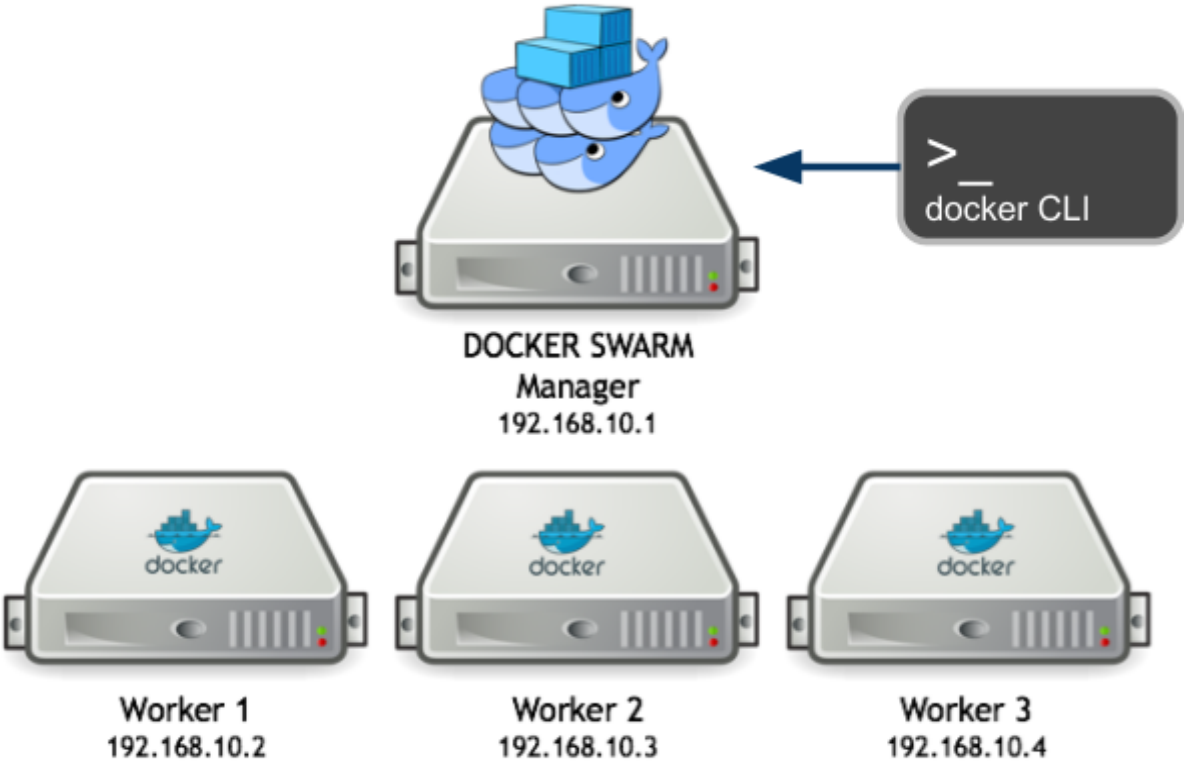
TP5 - Docker swarm / Kubernetes

Won't it be amazing to have a bunch of dockers hosts managed like a single one and to dispatch tasks to them in a simple way :)

https://docs.docker.com/engine/swarm/swarm-tutorial/create-swarm/

https://blog.docker.com/2016/07/docker-built-in-orchestration-ready-for-production-docker-1-12-goes-ga/

0 - Principles



https://sreeninet.wordpress.com/2017/08/15/docker-features-for-handling-containers-death-and-resurrection/

1 - Start a docker swarm manager

Applied to the CloudMIP platform, each user will start an instance featuring a public IP and having docker installed (see TP4 - Docker). Next is to configure this front-node as a **docker swarm manager**.

```
[root@cos75-ft ~]# yum -y install net-tools
[root@cos75-ft ~]# ifconfig
docker0: flags=4099<UP,BROADCAST,MULTICAST>  mtu 1500
      inet 172.17.0.1  netmask 255.255.255.0  broadcast 0.0.0.0
```

```
        ether 02:42:56:2a:8a:16  txqueuelen 0  (Ethernet)
        RX packets 0  bytes 0 (0.0 B)
        RX errors 0  dropped 0  overruns 0  frame 0
        TX packets 0  bytes 0 (0.0 B)
        TX errors 0  dropped 0 overruns 0  carrier 0  collisions 0

eth0: flags=4163<UP,BROADCAST,RUNNING,MULTICAST>  mtu 1450
        inet 192.168.0.169  netmask 255.255.255.0  broadcast 192.168.0.255
        inet6 fe80::f816:3eff:fe1a:ec15  prefixlen 64  scopeid 0x20<link>
        ether fa:16:3e:1a:ec:15  txqueuelen 1000  (Ethernet)
        RX packets 23914  bytes 32466192 (30.9 MiB)
        RX errors 0  dropped 0  overruns 0  frame 0
        TX packets 6260  bytes 817564 (798.4 KiB)
        TX errors 0  dropped 0 overruns 0  carrier 0  collisions 0

lo: flags=73<UP,LOOPBACK,RUNNING>  mtu 65536
        inet 127.0.0.1  netmask 255.0.0.0
        inet6 ::1  prefixlen 128  scopeid 0x10<host>
        loop  txqueuelen 1000  (Local Loopback)
        RX packets 0  bytes 0 (0.0 B)
        RX errors 0  dropped 0  overruns 0  frame 0
        TX packets 0  bytes 0 (0.0 B)
        TX errors 0  dropped 0 overruns 0  carrier 0  collisions 0

[root@cos75-ft ~]# docker swarm init --advertise-addr 192.168.0.169
Swarm initialized: current node (v44z77t96kklgrze0nihpa0zy) is now a manager.

To add a worker to this swarm, run the following command:

  docker swarm join \
  --token SWMTKN-1-14u71ujizez36h7lhdjjilg91fcrbnn72rra11gu6j86d43003-dmp53ta60h7cnd4h3suuz78bv \
  192.168.0.169:2377

To add a manager to this swarm, run 'docker swarm join-token manager' and follow the instructions.
```

Note: `docker swarm join-token worker` will remember you the command to join swarm

2 - Add docker hosts to the swarm

You now ought to launch 3 x COS75 m1.small instances that will automatically join your docker swarm manager.
https://docs.docker.com/engine/swarm/swarm-tutorial/add-nodes/

Your solution:

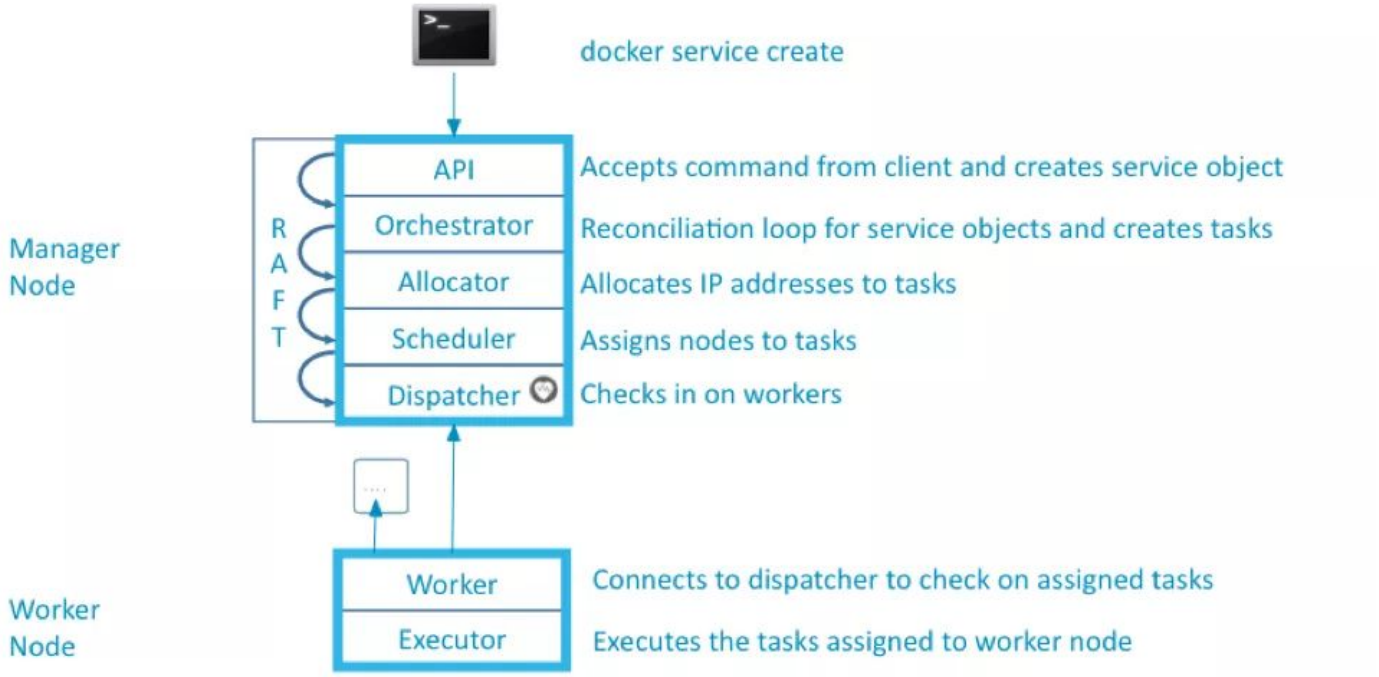
Hint: nova boot --min-count 3 --max-count 3 ...

Ok, to check that all of your CentOS docker hosts have joined your swarm manager:

```
[root@cos75-ft ~]# docker node ls
ID                            HOSTNAME                              STATUS  AVAILABILITY  MANAGER STATUS
nau6g9zkbqhw7e6vui35xmx2t     cos75-docker1-francois.novalocal  Ready   Active
xmx2tgtz6g9zkbqhw7e6vui35     cos75-docker2-francois.novalocal  Ready   Active
w7e6vuine5xmx2tu6g9zkbqh3     cos75-docker3-francois.novalocal  Ready   Active
v44z77t96kklgrze0nihpa0zy *   cos75-ft.novalocal                Ready   Active        Leader
```

3 - Launch a simple app

We'd like you to create a service across all of your docker hosts that belong to your swarm.
https://docs.docker.com/engine/swarm/swarm-tutorial/deploy-service/




```
docker service create --replicas 1 --name helloworld alpine ping cloudmip.univ-tlse3.fr

docker service ls

docker service inspect <service ID>

docker service ps
```

docker service create

- When creating a docker service, what's all this stuff about `--mode replicated` vs `global` ?

Your answer:

Hint: https://docs.docker.com/engine/reference/commandline/service_create/#options

- How will you launch a HTTP service on all of your docker hosts from your swarm ?

Your answer:

4 - Fedora Atomic

The Atomic project aims at providing the ability for Cloud platforms to become versatile enough to run docker containers. This objective is achieved through the availability of an image named 'Fedora atomic' that you launch like others instances but that only provides docker support.

http://www.projectatomic.io/docs/quickstart/

- Explore the various aspects of Fedora Atomic and try to bring out the major differences with regular Fedora image we've been using from the beginning.

- Exposes your recipe to build a Fedora atomic swarm.

5 - Bonus

- Docker vs **Singularity**: summarizes the key differences.

- How to share data across several VMs / containers ?
  Multiple answers possible

- Describe an infrastructure able to process Fabspace images. Each image to process will be hosted in a repository while processed images will be sent and stored in another directory. Fabspace images processing will involve the 'gimp' application.

- Now, you're a container manager and you ought to give root access to users inside a container … this means that your container needs to run both the user application along with a ssh daemon. Describe a dockerfile based on `supervisord` able to implement this behaviour.

Annexe-1 Openstack addon commands

We'll show simple use of the CloudMIP platform within the **service** project

create image from instance | nova snapshot

Creating a snapshot from a running instance

nova image-create --poll <instance_name> <snapshot_name>
nova image-create --poll private-instance COS75snap

… boot a new instance from this snapshot

nova boot --flavor *same_flavour* --image *<snapshot_name>* --nic net-id=c1445469-4640-4c5a-ad86-9c0cb6650cca --security-group default --key-name mykey <snapshot instance name>
nova boot --flavor m1.small --image COS75snap --nic net-id=c1445469-4640-4c5a-ad86-9c0cb6650cca --security-group default --key-name mykey myCOS75snap

Add Fedora atomic image

Search for Openstack atomic image
https://getfedora.org/en/atomic/download/

glance image-create --name FC29_atomic --disk-format=qcow2 --container-format=bare --visibility public --progress --file Fedora-AtomicHost-29-20181025.1.x86_64.qcow2

openstack image list
glance image-list

To add FC29 Openstack image (i.e featuring Cloud-init support)
https://getfedora.org/cloud/download/
… and repeat same process as above.

Persistent storage | cinder volumes

We'll add a persistent storage (cinder, i.e block) to an instance

cinder create --name FT_lv 4

Note: size in GB

nova list
cinder list *or* nova volume-list
nova volume-attach INSTANCE_NAME VOLUME_ID

```
nova volume-attach public-instance 1372f518-f06d-4ff5-9c3d-b31325ff3e51
+----------+--------------------------------------+
| Property | Value                                |
+----------+--------------------------------------+
| device   | /dev/vdb                             |
| id       | 1372f518-f06d-4ff5-9c3d-b31325ff3e51 |
| serverId | 49f31828-7ea8-444c-b518-6d8a957944ee |
| volumeId | 1372f518-f06d-4ff5-9c3d-b31325ff3e51 |
+----------+--------------------------------------+
```

log to VM [root only] direct SSH access to an instance and prepare newly attached storage

```
francois@fronta[~] sudo ip netns exec qdhcp-c1445469-4640-4c5a-ad86-9c0cb6650cca
ssh -i ~/.ssh/id_rsa fedora@192.168.0.153
Warning: Permanently added '192.168.0.153' (ECDSA) to the list of known hosts.
Last login: Thu Sep  8 11:23:12 2016 from 192.168.0.1
[fedora@private-instance-wn1 ~]$ sudo su
[root@private-instance-wn1 fedora]# fdisk -l
Disk /dev/vda: 20 GiB, 21474836480 bytes, 41943040 sectors
Units: sectors of 1 * 512 = 512 bytes
Sector size (logical/physical): 512 bytes / 512 bytes
I/O size (minimum/optimal): 512 bytes / 512 bytes
Disklabel type: dos
Disk identifier: 0x06d4f68c

Device     Boot Start      End  Sectors Size Id Type
/dev/vda1  *     2048 41943039 41940992  20G 83 Linux

Disk /dev/vdb: 1 TiB, 1099511627776 bytes, 2147483648 sectors
Units: sectors of 1 * 512 = 512 bytes
Sector size (logical/physical): 512 bytes / 512 bytes
I/O size (minimum/optimal): 512 bytes / 512 bytes
```

- *create a partition /dev/vdb1*
- *dnf -y install **xfsprogs***
- *mkfs.xfs /dev/vdb1*
- *mount newly created partition … et voilà :)*

Annexe-2 Docker commands

Functional test

```
docker info
docker run hello-world
```

	docker
Links	http://reseau-loops.github.io/presentations/2015-12-17-sebastien-binet-docker.pdf [Docker] setup for ...
ps	docker ps → list running dockers docker ps -a → list ALL dockers (even those in a stopped state)
commit	*you added / modified a running container → commit changes to images* **docker commit -m 'update'** <containerID> <image_name:tag> *docker commit 5fab36bbec1e sensocampus/django:latest*
disconnect	**Ctrl p Ctrl q** → this will disconnect from a connected container
attach	*Attach to a running docker you've disconnected from* **docker attach <containerName>**
exec	*execute a command within a running container* **docker exec -it <container_name> bash**
run + entry	*to run a specific entry point for a docker* **docker run -it --entrypoint bash mqttocampus**
link	*Superseded by docker-compose* *Link to an existing container from an interactive docker* docker run --rm -t -i --link <dockerName>:<aliasOpt> <dockerImg> bash **docker run --rm -t -i --link mysql-idex mysql bash**
logs	*view containers' logs (trapped **stdout** and **stderr**)* **docker logs -ft <container_name>** *view journalctl entries from host* **journalctl -ef CONTAINER_NAME=<container_name>** *Note: it is '-ef', not '-ef**u**'*
volume	*create, list, inspect volumes* **docker volume create -d local --name mysql-idex -o size=20G** *(it seems that size does not matter with this driver)*

attach dir / volume	*attach a host directory* **docker run -v <host_dir>:<container_dir> ...** *attach a volume* **docker run -v <vol_name>:<container_dir> ...**
network	*list network resources* **docker network ls** **docker network inspect bridge**
volumes-from	*Attach a volume from a container dedicated as volume holder* TODO
launch with command	*to launch a specific command from an image: run shell from alpine linux* **docker run -it --rm alpine sh**
rename (tag)	*to rename a image (is only a link)* **docker tag '*<imageID>*' *name:tag***
build	**docker build --no-cache -t *name:tag* -f *mydockerfile* .**
enter a build failed container	*a container failed to build … how to enter within ?* **docker commit `docker ps -q -l` failure** **docker run -it failure /bin/bash**
save / load images	*save docker image(s)* **docker save -o neocampus.tar mqttOCampus:latest sensocampus/django:latest ...** *load saved docker images* **docker load -i neocampus.tar**
save / load volumes	*save a data volume* **docker stop mongo-neOCampus** **docker run --rm -v *volumename*:/vol -w /vol alpine tar -c . > *volume*.tar** *load saved data volumes* **docker run --rm -v *volumename*:/vol -w /vol -i alpine tar -x < *volume*.tar**
copy files from container to local dir	**docker cp <container>:/<remote_dir> <local_dir>** *e.g* **docker cp mongo-neocampus:/dump .**

[DEPRECATED] docker config files

Please make use the **new** `'/etc/docker/daemon.json'` config file … these are for reference only!

- `/etc/sysconfig/docker`

.....

```
OPTIONS='-g /dockyard --log-driver=journald --signature-verification=false'
.............
```

- `/etc/sysconfig/docker-network`

```
# /etc/sysconfig/docker-network
DOCKER_NETWORK_OPTIONS="--mtu=1450 --bip=172.17.0.1/24
--fixed-cidr=172.17.0.0/24"
```