Master2 ILORD | Virtualisation

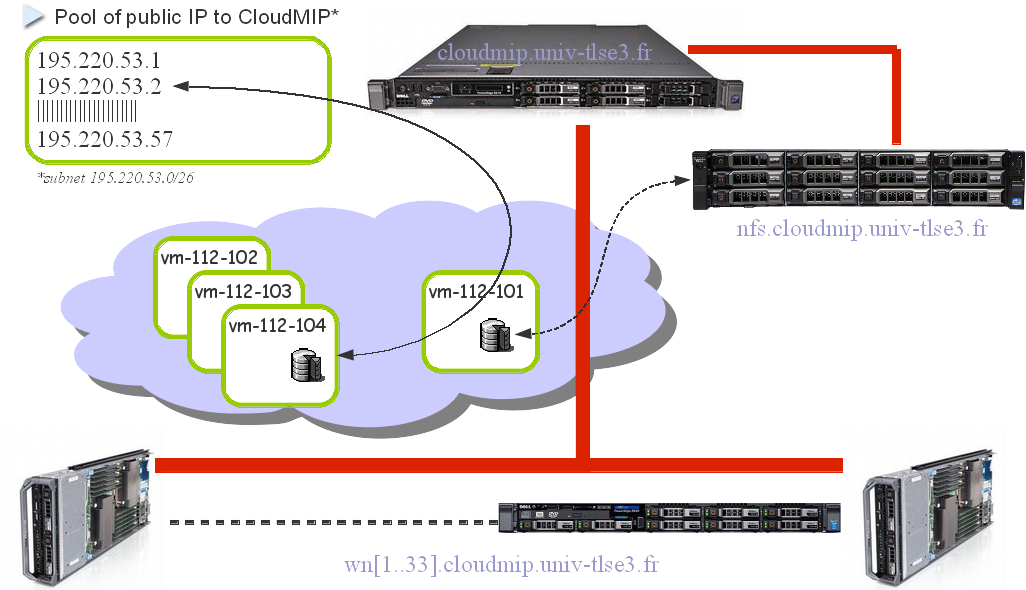
Virtualisation practical exercises

Dr Thiebolt François



# TP1 - Basic instances | the GUI way

We'll start with the deployment of some basic instances.To achieve this, we will first start with the Horizon GUI, next with CLI.



Here is the main architecture of the CloudMIP platform.

Please have a look to our Wiki for a more detailed hardware description:

<http://cloudmip.univ-tlse3.fr/platform/platform>

## 1 - Horizon GUI

Using your freshly acquired credentials, you will now connect to

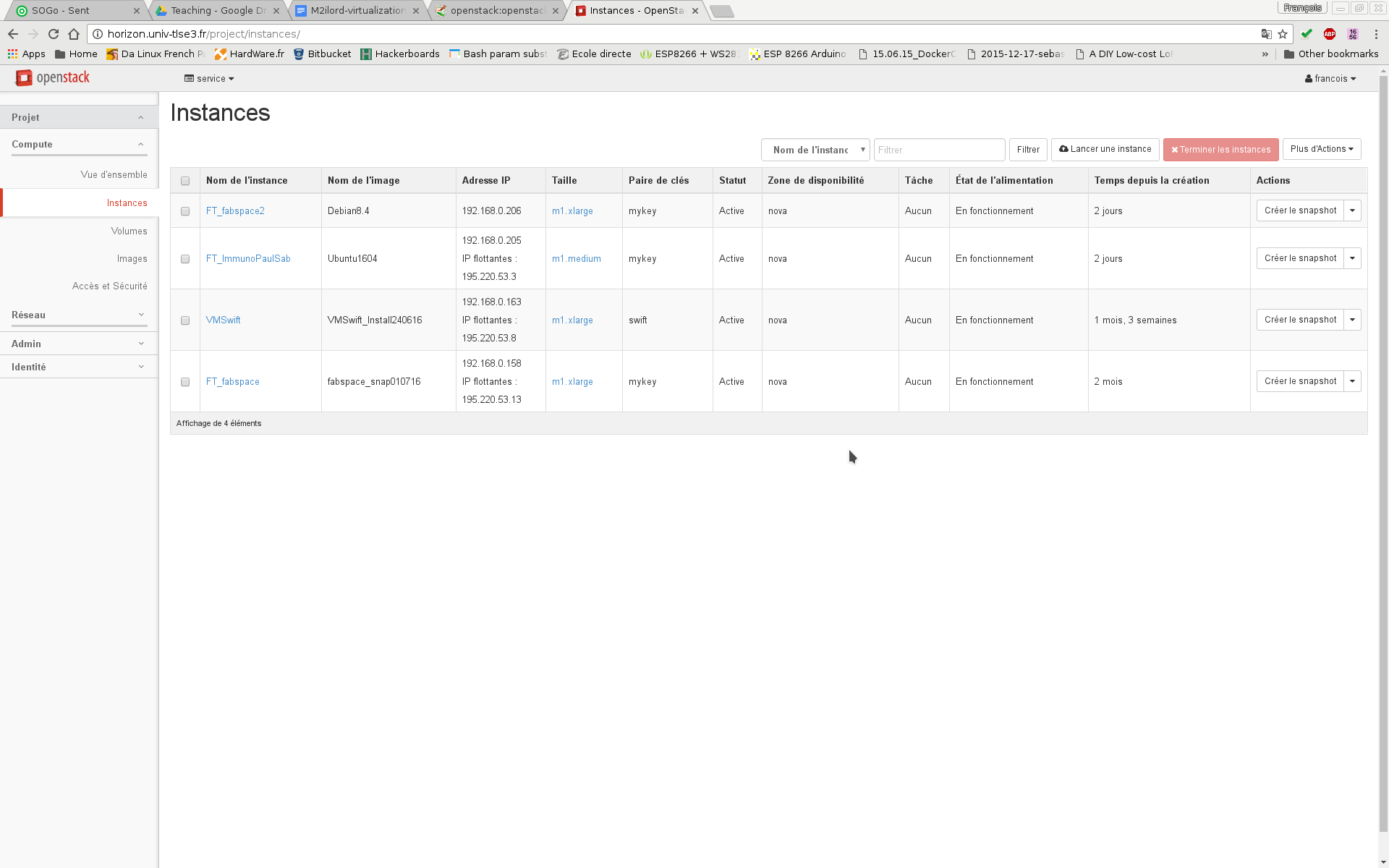
|  |
| --- |
| [http://horizon.univ-tlse3.fr](http://horizon.univ-tlse3.fr/) |

Our first step will be to create a public / private key pair:

*Projet → Compute → Access et Sécurité → Créer une paire de clés*

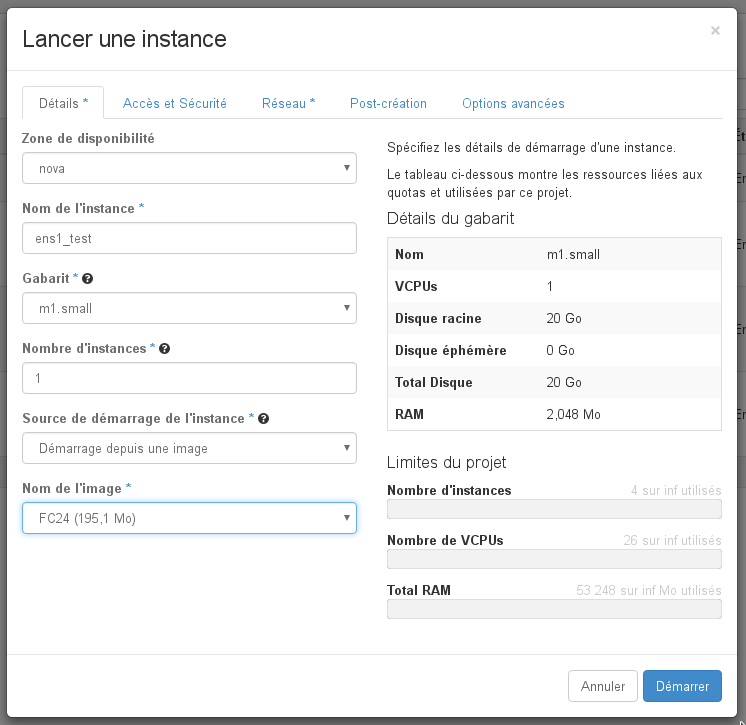


Then you continue to the instance creation form *Projet → Compute → Instances*



Now you go to the *Compute* tab and you select '**Lancer une instance**'

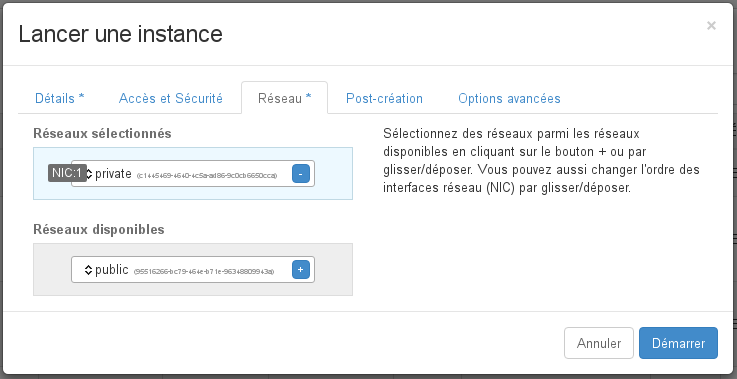
*(You choose parameters according to the screenshoots*.)



Select **default** as security group in *Accès et Sécurité*



In the *Network* tab, add a **private IP**



… and click 'Start'!

OK, you now have launched an instance featuring a **private IP** …that's great … but hey!?! … how do i get connected to my VM ??? → **you can't!**

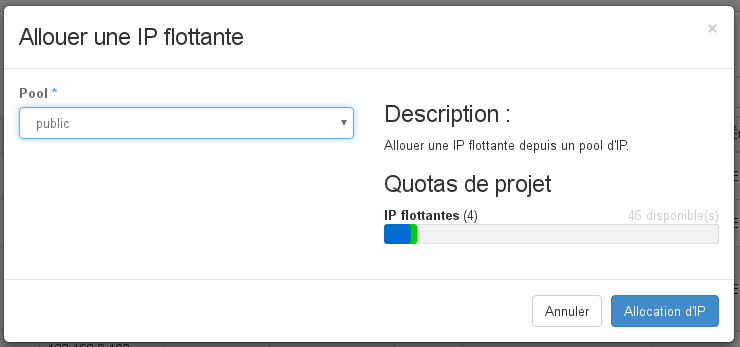
(neither VNC nor direct connection through internal network ---linux network namespaces prevent this! … this is not a flat network)

So, to get connected to your VM … you need to add a **Public IP**

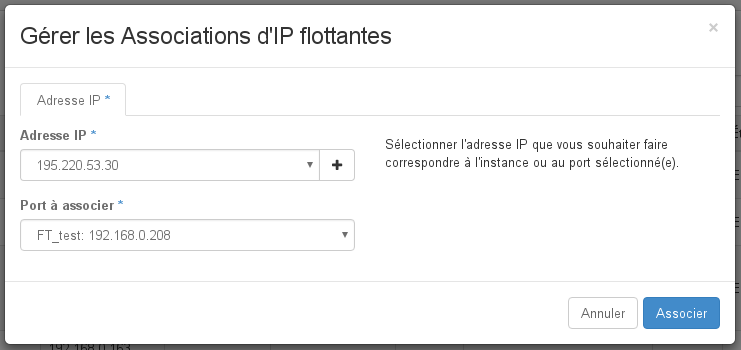
*Project → Compute → Instance → Action (from your instance row) → Associer une IP flottante*



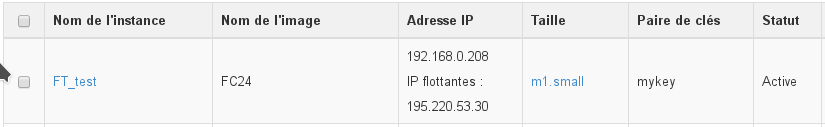
you then click **+**



… then '**Allocate IP**' ...



....and finally '**Associate**'



Your instance now features both a public and a private IP … let's get connected!

### First connection to your VM | SSH

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> fedora@<public IP> |
| ssh fedora@195.220.53.30 |

you then change root passwd

|  |
| --- |
| [fedora@ft-test ~]$ sudo su -  [root@ft-test ~]#passwd |

Note: take care of cleaning a bit ~/.ssh/authorized\_keys file from your root account

|  |
| --- |
| echo "fastestmirror=True" >> /etc/dnf/dnf.conf |

you can now install some additional software

|  |
| --- |
| dnf -y install nginx  systemctl enable nginx  systemctl start nginx |

now you can check with http://<your\_public\_IP>

you can also check what's your IP from abroad

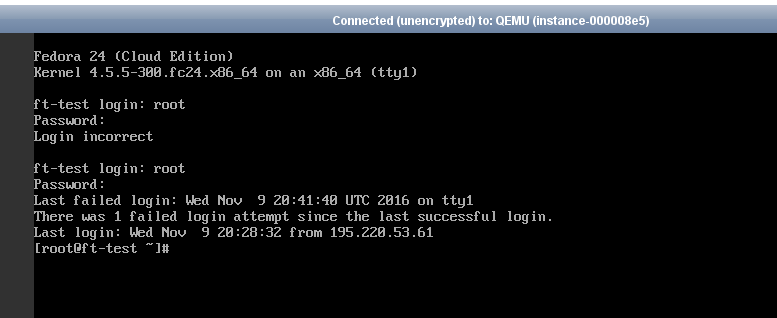
|  |
| --- |
| dnf -y install bind-utils |

|  |
| --- |
| dig +short myip.opendns.com @resolver1.opendns.com |
| curl ipinfo.io/ip |

### 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!*

## 2 - Others Actions | snapshot

At this time, you can now create a **snapshot** of your running instance.

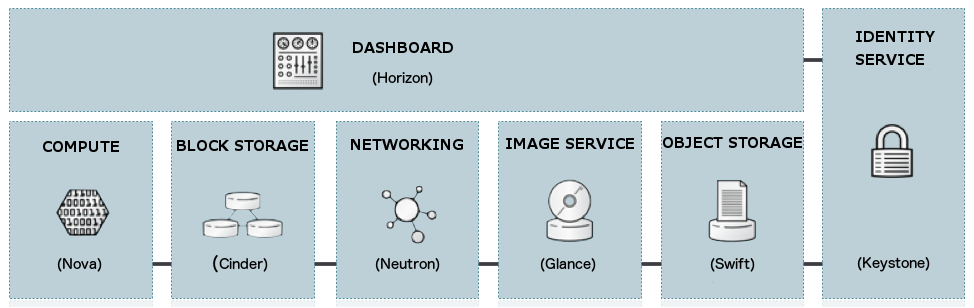
(this is not live snapshotting i.e it will stop your instance and restart it)

You now terminate your instance and **start a new one** from your newly created snapshot.

**>>> don't forget to stop (i.e destroy) running instances when you're done <<<**

# TP2 - Basic instances | the CLI way

We have had fun with the GUI … now let's exercises our freshly acquired knowledge with 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 execute command that will prepare start files,

|  |
| --- |
| osSetCredential.sh |

WARNING: this command will create env. var OS\_PASSWORD with your password in your ~/.env file → ensure that no one else can access it! As an alternative, have a look at *service-openrc.sh* *in* [*TP3 - pre-requisites*](#_wwuuf9a5vv3)

generate a public/private keypair,

|  |
| --- |
| ssh-keygen -t rsa |

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 |

Note: feel free to delete any previously existing SSH keys → nova keypair-delete <key\_ID>

## 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

We'll now launch an instance of the FC24 image.

|  |
| --- |
| 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 ought to use.

|  |
| --- |
| nova boot --flavor m1.small --image **FC24** --nic net-id=*<NETWORK\_ID>* \ --security-group default --key-name mykey *instance\_name* |
| nova boot --flavor m1.small --image **FC24** --nic net-id=c1445469-4640-4c5a-ad86-9c0cb6650cca --security-group default --key-name mykey *myinstanceName* |

|  |
| --- |
| nova list |

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

|  |
| --- |
| nova show *<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 are no 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>* **195.220.53.4** |

### Connect to your fedora VM

|  |
| --- |
| ssh **fedora**@195.220.53.4  [fedora@private-instance ~]$ sudo su -  [root@private-instance ~]# |

*... well done player one ;)*

**>>> don't forget to terminate instances and to free allocated IPs (nova floating-ip-delete <IP>) <<<**

## 

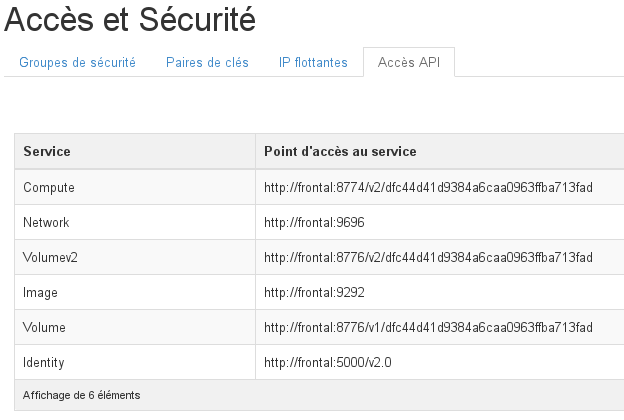
# TP3 - Basic instances | the API way

We'll now launch instances using the Openstack APIs.

As a first step, you ought to retrieve the various Openstack API services end-point.

Log with your account to [http://horizon.univ-tlse3.fr](http://horizon.univ-tlse3.fr/)

*Projet → Compute → Accès et sécurité → onglet Accès 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 |

## 

## 0 - Foreword | HTTP calls

Before going on with python APIs, we give here some HTTP calls examples.

To ease interactions with API, we define some environment variables that you should use in an interactive bash shell or save as a script:

|  |
| --- |
| export OS\_PROJECT\_ID="dfc44d41d9384a6[caa0963ffba713fad](http://frontal:8774/v2/dfc44d41d9384a6caa0963ffba713fad)"  export OS\_COMPUTE\_API="http://frontal:8774/v2/${OS\_PROJECT\_ID}"  export OS\_NETWORK\_API="[http://frontal:9696](http://frontal:9696/)"  export OS\_STORAGE\_API="http://frontal:8776/v2/${OS\_PROJECT\_ID}"  export OS\_IMAGE\_API="[http://frontal:9292](http://frontal:9292/)" |

*Note: OS\_AUTH\_URL is already set from /etc/profile.d/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 | 2016-11-14T20:04:58Z |  | id | **fbc84bccd8f74e60af059ef83a5e7220** |  | project\_id | dfc44d41d9384a6caa0963ffba713fad |  | user\_id | xxx |  +------------+----------------------------------+ |

… then define env. var with obtained token\_id

|  |
| --- |
| export OS\_TOKEN="**fbc84bccd8f74e60af059ef83a5e7220**"  export OS\_URL="${OS\_AUTH\_URL}" |

*Note: when using* ***tokens****, you must provides OS\_URL env. var. for CLI commands*

We can now explore some of the various OpenStack APIs.

<http://developer.openstack.org/api-guide/quick-start/api-quick-start.html>

To list images available for launching instances

|  |
| --- |
| curl -s -H "X-Auth-Token: $OS\_TOKEN" ${OS\_COMPUTE\_API}/images | python -m json.tool |

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 |

<http://developer.openstack.org/api-ref/compute/>

## 1 - Python API to list running instances

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

* list running instances (i.e servers) and all of their IPs.

To achieve this, we'll make use of the Nova python client and previously defined environment variables

<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>

* myInstancesList.py

|  |
| --- |
| #!/usr/bin/env python  # coding: utf-8  # Import zone  #  import errno  import os  import signal  import syslog  import sys  import time  import threading  import json  import logging  # CLI options  from optparse import OptionParser  # OpenStack  from keystoneauth1.identity import v2  from keystoneauth1 import session  #from keystoneclient.v2\_0 import client  from novaclient.client import Client as novaClient  # Global Variables  #  \_shutdownEvent = None # signal across all threads to send stop event  # Function ctrlc\_handler  def ctrlc\_handler(signum, frame):  global \_shutdownEvent  print("\n<CTRL + C> action detected ... ");  assert \_shutdownEvent!=None  \_shutdownEvent.set()  # Main function  def main():  # Global variables  global \_shutdownEvent  # create threading.event  \_shutdownEvent = threading.Event()  # Trap CTRL+C (kill -2)  signal.signal(signal.SIGINT, ctrlc\_handler)  # Parse CLI arguments  parser = OptionParser()  parser.add\_option("-d", "--debug",  action="store\_true", dest="debug", default=True,  help="Debug mode")  (options, args) = parser.parse\_args()  #  # authentication  token = None  auth = None  sess = None  # Does OS\_TOKEN exists ?  token = os.getenv('OS\_TOKEN', None)  if token is not None:  print("Using env. OS\_TOKEN found (%s)" % token);  try:  #**TODO**: implement token based authentication  except Exception as ex:  print("###ERROR obtaining password authentication against '%s'" % os.getenv('OS\_AUTH\_URL'))  raise Exception(ex);  else:  print("No token found, trying to use username / password");  try:  auth = v2.Password(auth\_url=os.getenv('OS\_AUTH\_URL'),  username=os.getenv('OS\_USERNAME'),  password=os.getenv('OS\_PASSWORD'),  tenant\_name=os.getenv('OS\_TENANT\_NAME'));  except Exception as ex:  print("###ERROR obtaining password authentication against '%s'" % os.getenv('OS\_AUTH\_URL'))  raise Exception(ex);  # check authentication mechanism ...  if auth is None:  raise Exception("Unable to authenticate ... aborting :(");  # launch session ...  print("Starting session ...");  try:  sess = session.Session(auth=auth)  except Exception as ex:  raise("Unable to start session ... :(\n" + ex)  # check session  if sess is None:  raise("Unable to establish session .... aborting :(");  #  # clients  nova = None  # start nova client  print("\t... start nova client ...");  try:  #nova = client.Client("2.1", session=sess)  nova = novaClient("2", session=sess)  except Exception as ex:  print("Unable to start nova client : " + ex)  sys.exit(1);  #print(nova.floating\_ips.list())  #  # main loop  while not \_shutdownEvent.is\_set():  #print(nova.servers.list())  #**TODO**: list all servers along with their IP  \_shutdownEvent.wait(2)  # Execution or import  if \_\_name\_\_ == "\_\_main\_\_":  main() |

|  |
| --- |
| chmod a+x ./myInstancesList.py  ./myInstancesList.py |

## 2 - Start / stop instance

Next, your python application myStartStop.py would have to:

* start a simple instance (i.e no public IP, only a private one),
* destroy instance.

<http://docs.openstack.org/developer/python-glanceclient>

## 3 - Get connected to your instance

Going one step further, your python application myInstanceManager.py will:

* start an instance,
* allocate and associate a public IP,
* get SSH connected\* to your instance to launch a simple command (e.g hostname)
* detach and destroy allocated public IP
* clear allocated resources (i.e delete instance)

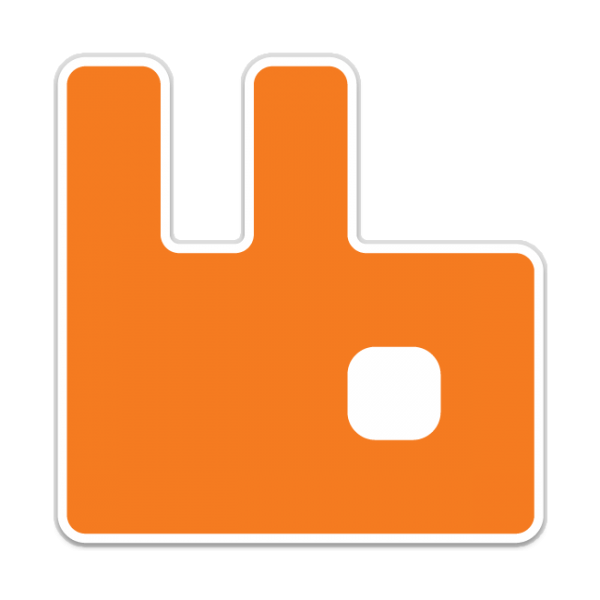
*\*paramiko is a python SSH client module*

<http://sebastiandahlgren.se/2012/10/11/using-paramiko-to-send-ssh-commands/>

# TP4 - Workload dispatcher

Previously, we had explored various ways (GUI, CLI and API) to launch a single instance (i.e a server): it's now time to go ahead!

In this scenario, we propose you to implement a job scheduler featuring an elastic behaviour: the more tasks are being collected in the processing queue, the more you allocate instances to process them.



All of the instances your dispatcher may start will feature the same capabilities (1 VCPU and 2GB RAM ---m1.small)

To leverage our inter-nodes communication needs, we'll make use of **RabbitMQ**.

Public network

Private network

Processing queue

## 1 - RabbitMQ communication queues

To reach our goal, we'll need to have network-enabled messages queue. On one side of this queue, we'll have a **producer**; on the other side, we'll have up to **MAX\_CONSUMERS**.

The producer add tasks to the queue while consumers will grab and process tasks from the queue. **MAX\_JOBS\_QUEUE** is the maximum number of jobs waiting in the queue.

As a first step, you may consider jobs as being a random time to wait for (e.g 10s to 60s)

Shell command to sleep for a random amount of time

|  |
| --- |
| sleep $(shuf -i 10-60 -n1) |

Before going on with the development of your python workload dispatcher, you first need to:

Start your public instance

|  |
| --- |
| *start an instance with a private and a public IP* |

Log as root in your public instance and start **RabbitMQ** setup:

|  |
| --- |
| echo "fastestmirror=True" >> /etc/dnf/dnf.conf  dnf -y update  dnf -y install rabbitmq-server  dnf -y install python3-pika  systemctl enable rabbitmq-server  systemctl start rabbitmq-server |

add user(s) to RabbitMQ

|  |
| --- |
| rabbitmqctl add\_user master cloudmip rabbitmqctl set\_permissions master ".\*" ".\*" ".\*" rabbitmqctl set\_user\_tags master administrator  rabbitmqctl list\_users |

It's now time to start to develop your python workload dispatcher myWorkloadDisptacher.py.

Going one step after the other, you may:

* use **pika** python3 client to create a queue (**MAX\_QUEUE\_LENGTH**) and to connect to,
* add methods to create and push messages in the work queue,
* create a thread that will consume messages from the work queue.

To help you, have a look to the various links below

<https://www.rabbitmq.com/tutorials/tutorial-one-python.html>

<https://www.rabbitmq.com/tutorials/tutorial-two-python.html>

**>>> DON'T FORGET TO SAVE YOUR python code in your CloudMIP account too! <<<**

Make a snapshot of your public instance

|  |
| --- |
| nova image-create --poll *<your public instance\_name> <snapshot\_name>* |

***Note****: don't forget to prefix with your id (e.g ens1) all of your snapshots, instances etc*

Stop your public instance

|  |
| --- |
| nova delete *<your public instance\_name>* |

## 2 - one public instance to rule them all

At this point, you now have a snapshot of an instance featuring a RabbitMQ server and a python3 AMQP client.

* start a public instance from your snapshot,
* start two additional instances with just a private IP for each of them

Manage SSH keys to enable seamless SSH remote execution from public instance to private ones.

|  |
| --- |
| (from public instance)  ssh root@*<private instance>* date |

## 3 - static workload dispatcher

Having now both a public and two private instances running, it's now time to expand your python code in the following way:

* myWorkloadDisptacher.py application must have support for CLI options like -p (producer) or -c (consumer),
* Producer may generate messages containing tasks to execute.
* launched with -c in a consumer instance, your application will have to retrieve message (job) from work queue and start to process it.

Producer may not create & push more than **MAX\_JOBS\_QUEUE** jobs in the work queue.

## 4 - elastic Workload dispatcher

In this final step, our producer will generate jobs using a gaussian function.

As the number of jobs increase in the work queue, your python workload dispatcher (producer mode) will start to instantiate additional instances (up to **MAX\_CONSUMERS**) that will consume the tasks waiting for processing in the work queue.

Once the number of tasks waiting to get processed decrease, your python workload dispatcher will stop instances in order to match the workload.

You ought to pay attention to the fact that launching an instance takes some time. It means that: to avoid flickering effect you ought to add some hysteresis in your create / destroy instance method.