LAB – OpenStack basic operations

Hands on experience with OpenStack basic operations

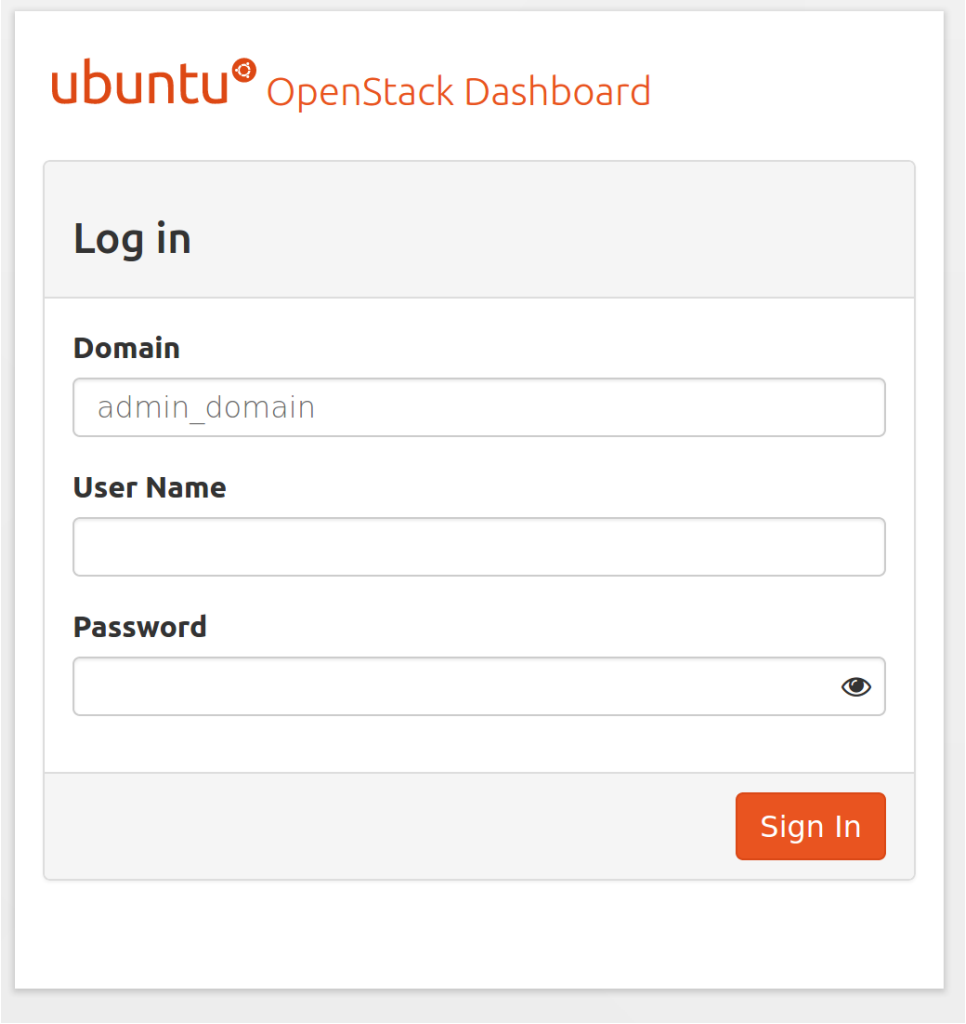
References:

- OpenStack documentation

Connect to the dashboard

- Open the browser to the URL:
  - http://CONTROLLER_IP/horizon/auth/login/

- Domain: admin_domain
- User Name: admin
- Password: openstack



Finalize the configuration

- A set of configurations are still missing for the platform to be functional
  - Network configuration
  - Flavors configuration
  - Image configuration

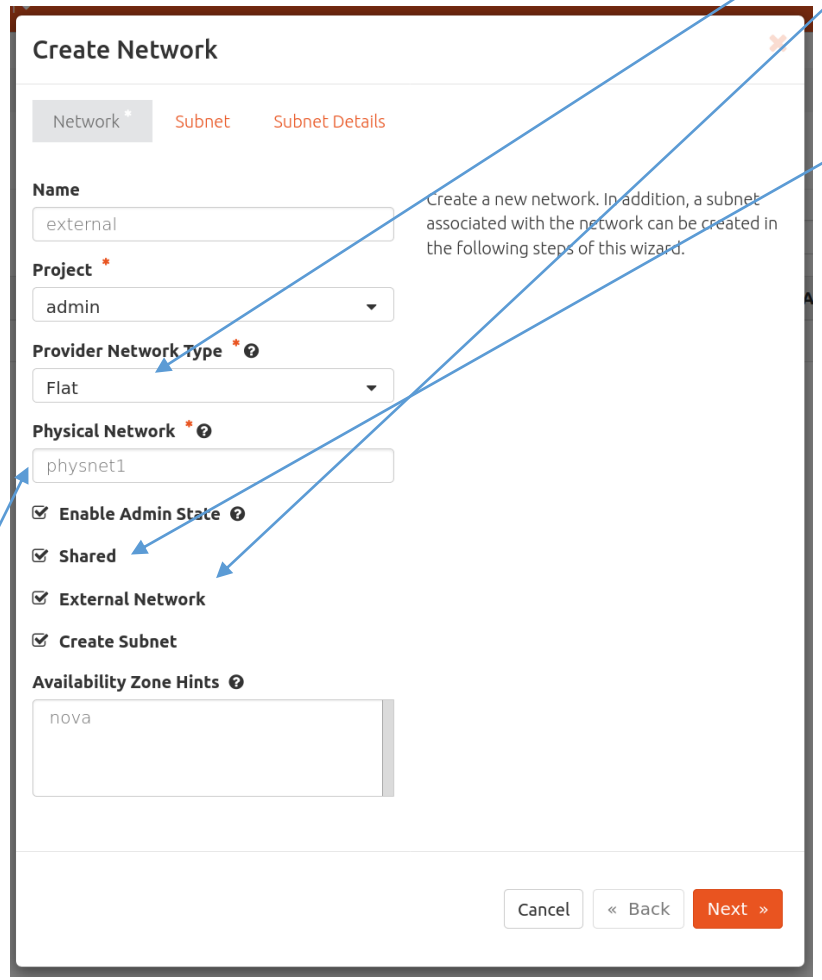
Network configuration

- A final configuration of the network is needed for the installation to be operational
- Specifically at least a couple of virtual networks has to be created to connect VMs among themselves and to connect the VMs to an external network (and eventually the Internet)
- The networks we want to create are:
  - An **external** network that is linked with a real external network to connect VMs with the outside world
  - An **internal** network (virtual) to link the VMs
- In general VMs will have a virtual NIC connected to the internal network, a router will be deployed to link the VMs with the external networks (through NAT as we will see)

Create the external network

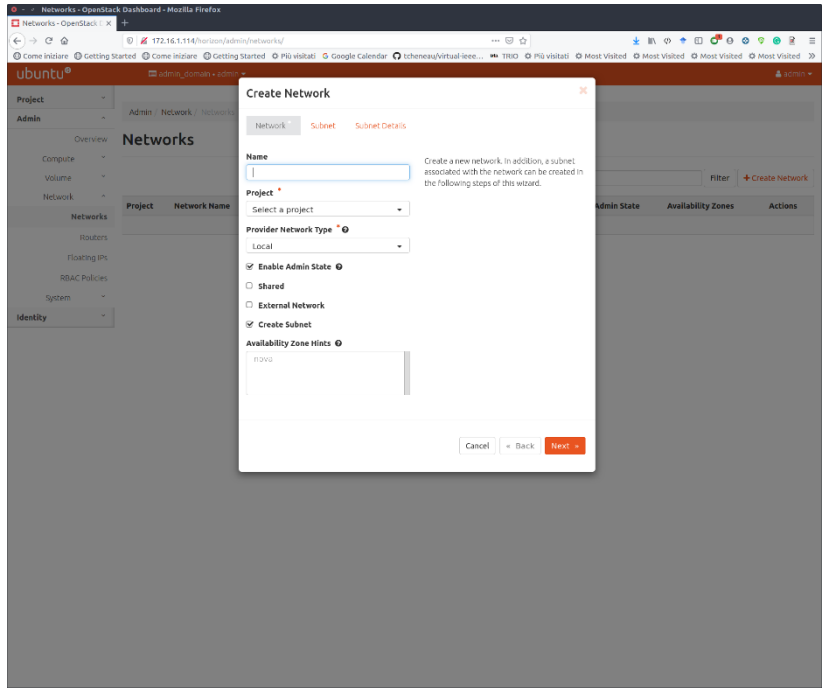
- Admin -> Network -> Networks -> Create Network
- Create a new network with the following params:

This subnet is the external network, it will be bridged with physnet1 linked to vlan2 (as configured during installation)



The type of network is flat, as it will be linked to a physical network

The network will be shared among all the users of the platform

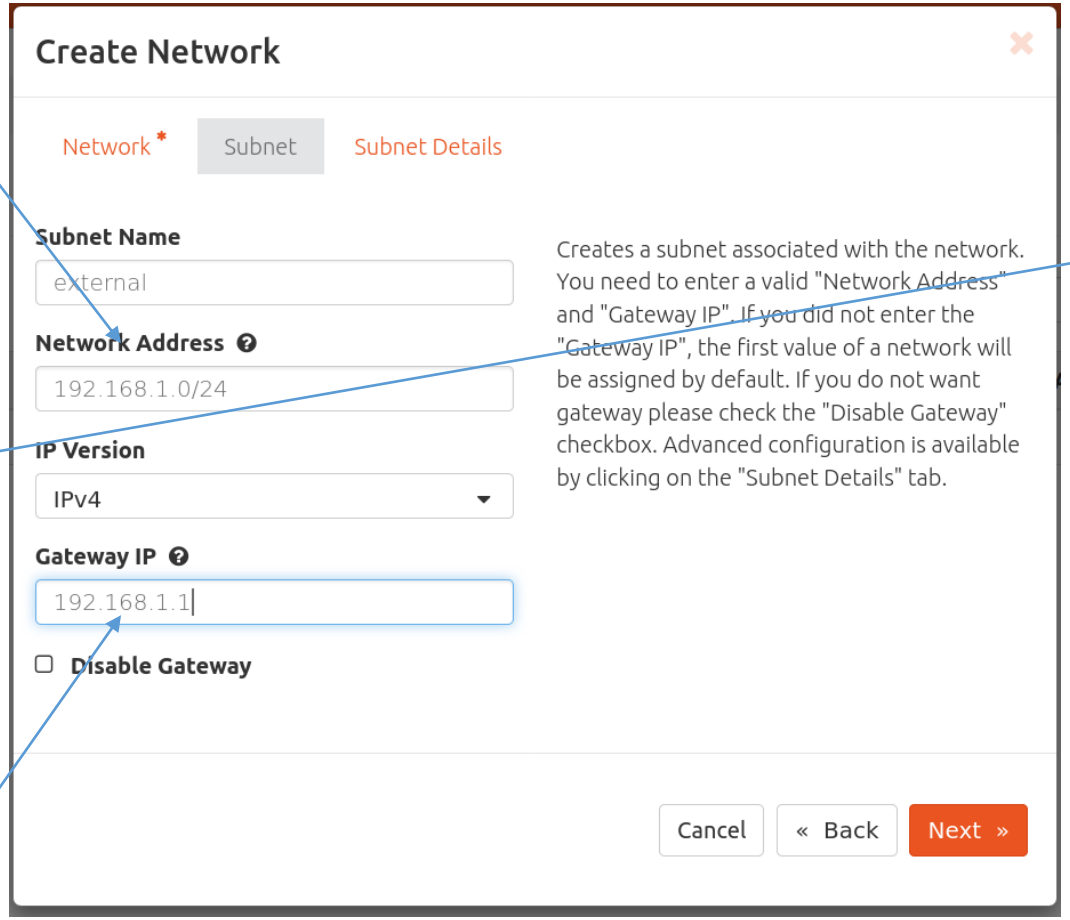


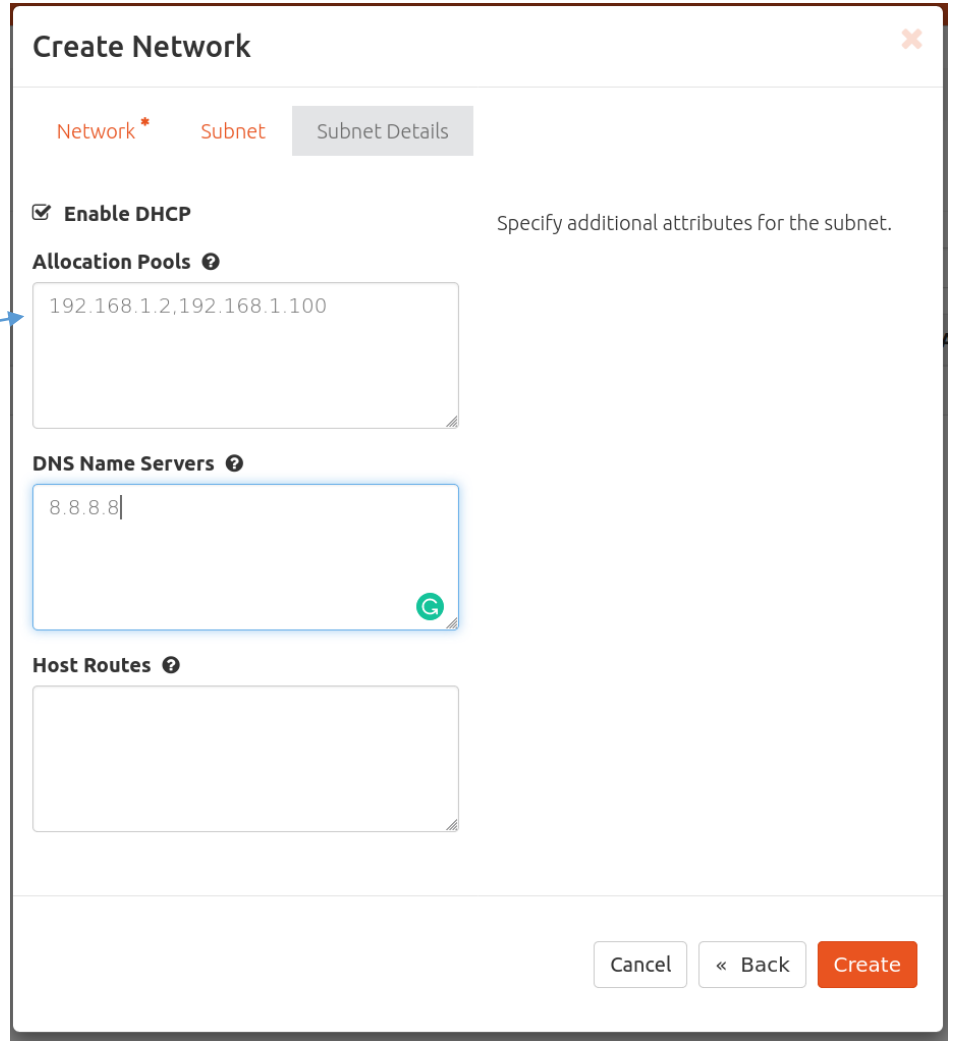
Create the subnet for the external network

This is the address of the network

This pool of IPs will be available to VMs that require external connection

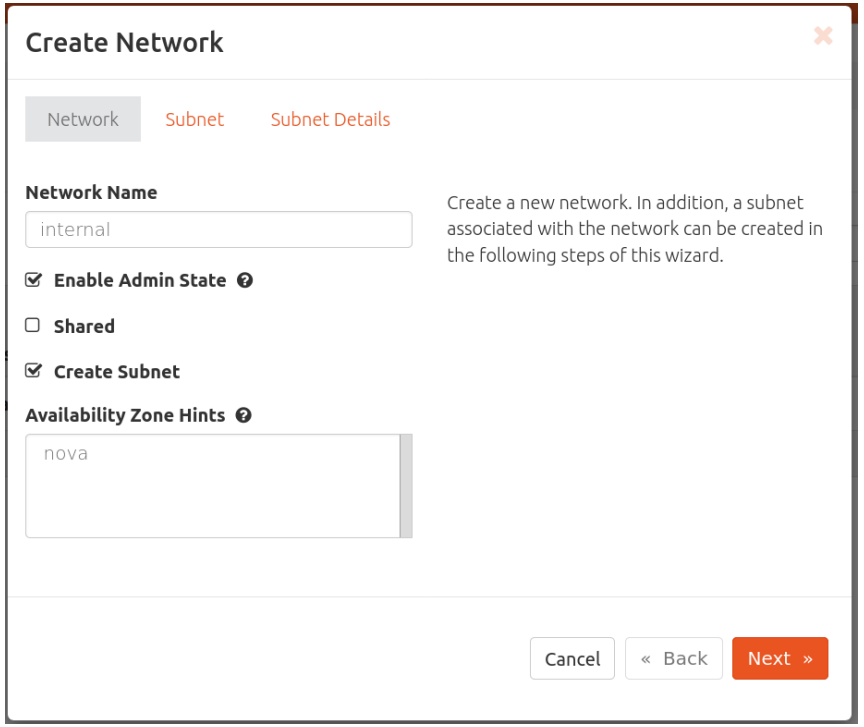
This is the IP of the gateway. It is supposed to be a real gateway on the network (we will make one)



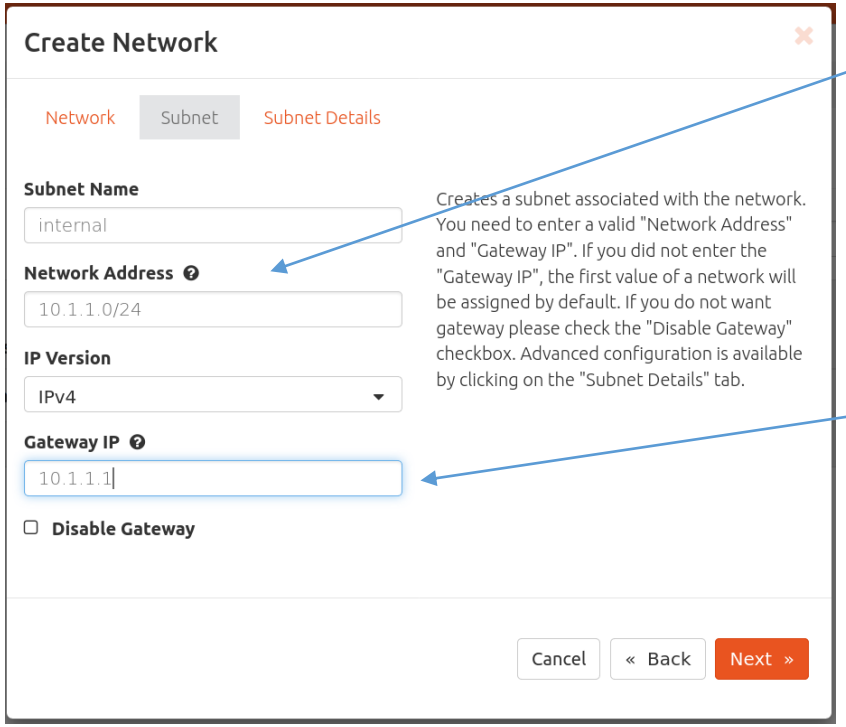


Create an internal network

- Create an internal network for the VMs created by the admin user (each user of the platform can have its own virtual network)
- Project -> Network -> Networks -> Create Network

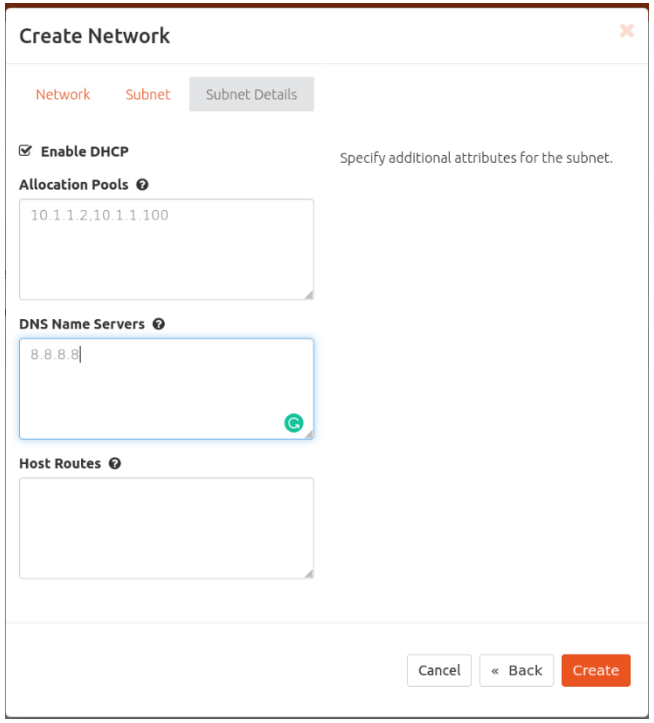


Create an internal network



This network will be the virtual one hosting the VMs

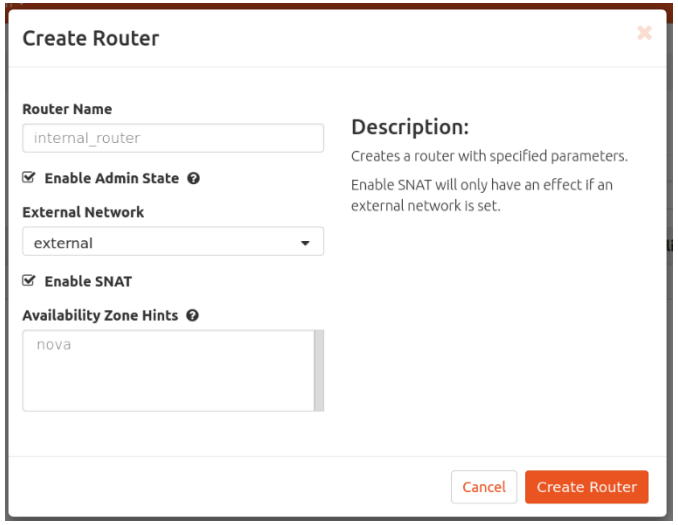
This gateway will be a virtual gateway



Create a virtual router for the internal net

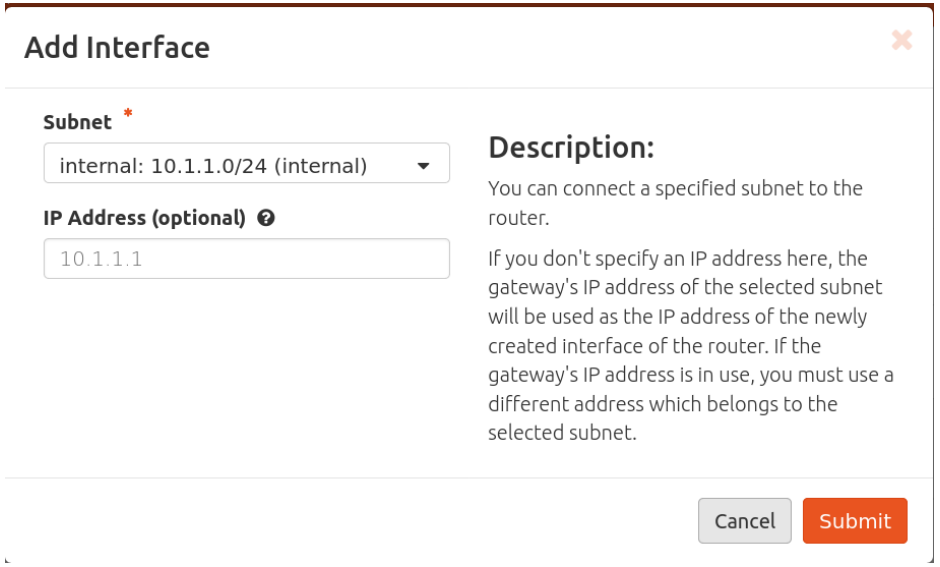
- Create a virtual router or gateway to link the virtual network with the external one
- Project -> Network -> Routers -> Create Router

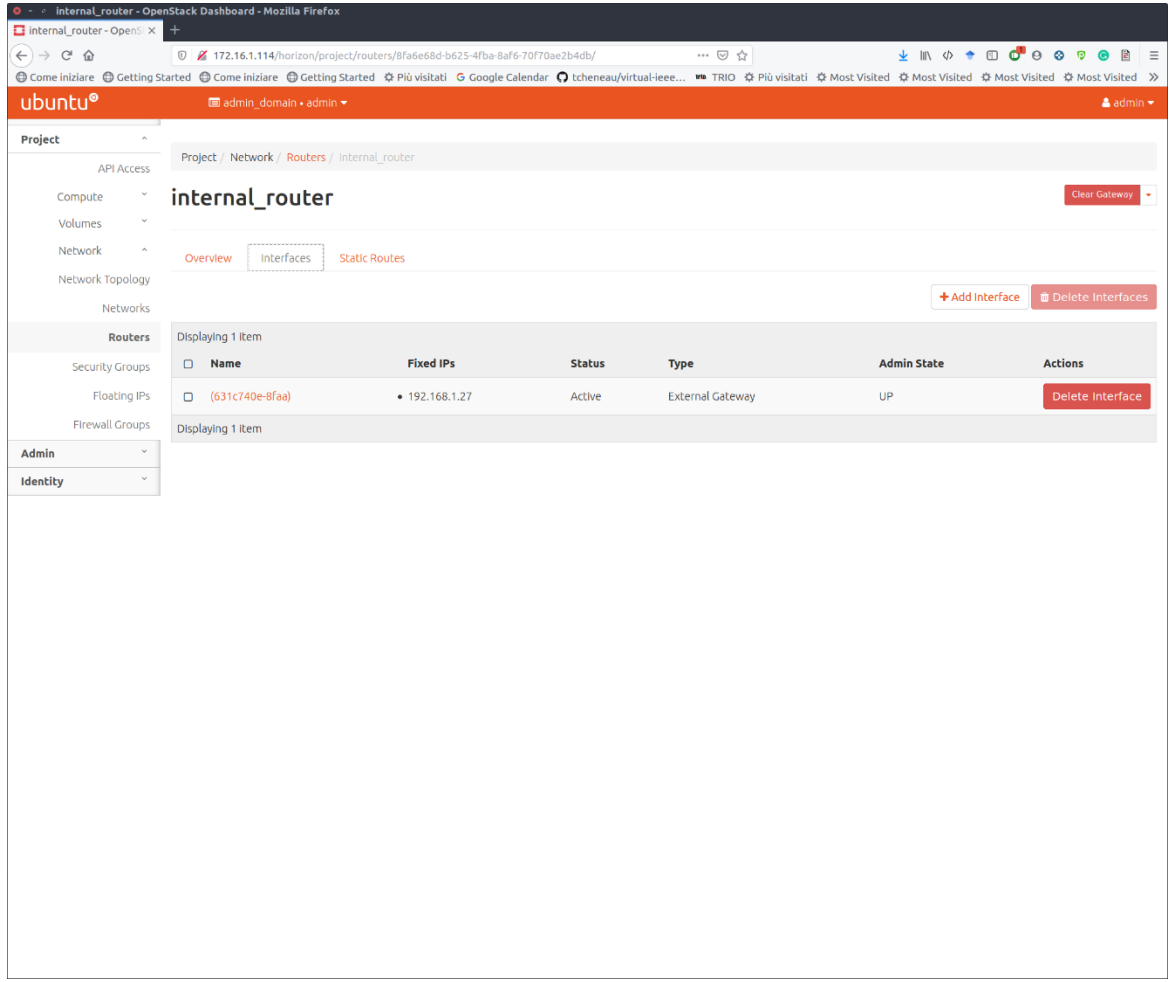
The router will have an interface on the external network



Connect router to the internal network

- The router needs to be explicitly connected to the internal network
- Enter in the router configuration, interfaces tab
- Then Add Interface:



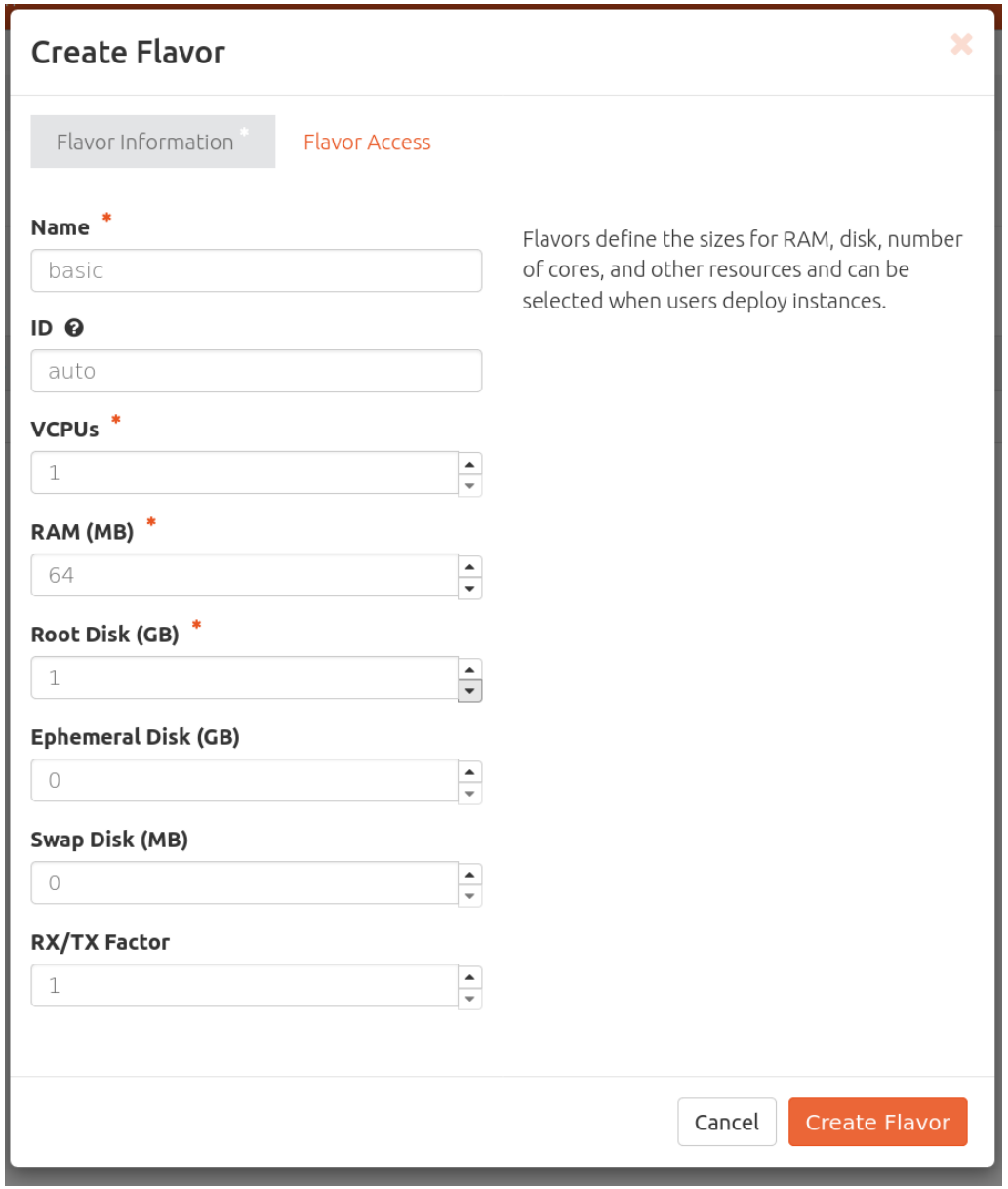


Network configuration

- The network configuration is completed, and it has the following topology

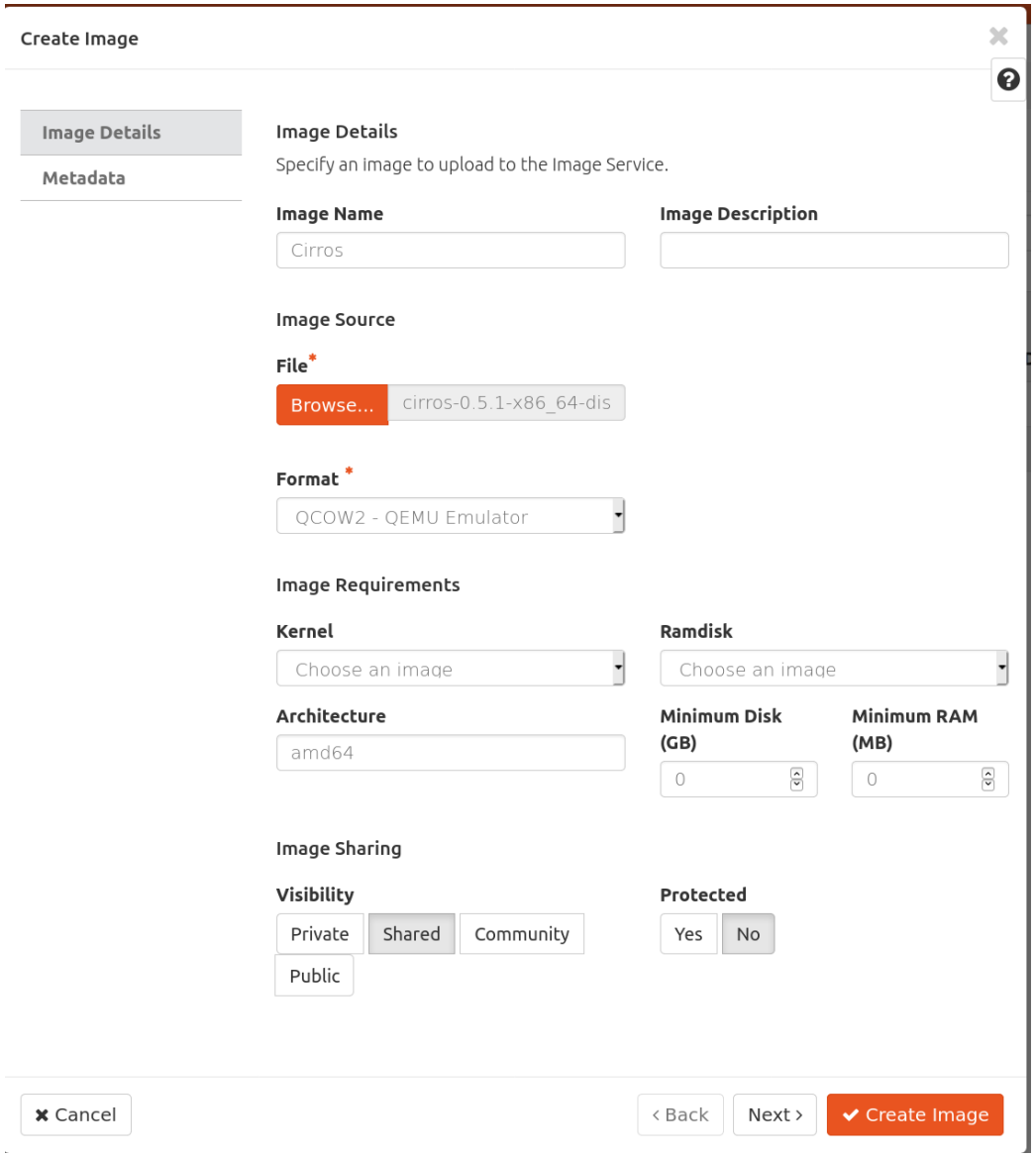
Flavors

- A flavor is a configuration for a virtual machine
- It defines the amount of resources that are allocated to a VM at the time of creation
- The administrator must create at least one flavor
- Admin->Flavors->Create Flavor



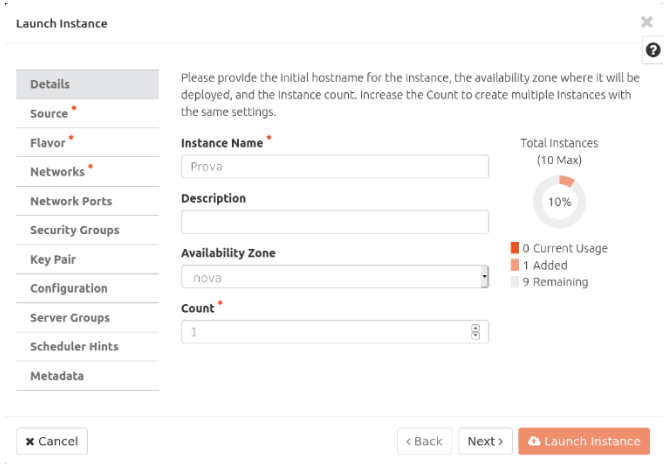
Images

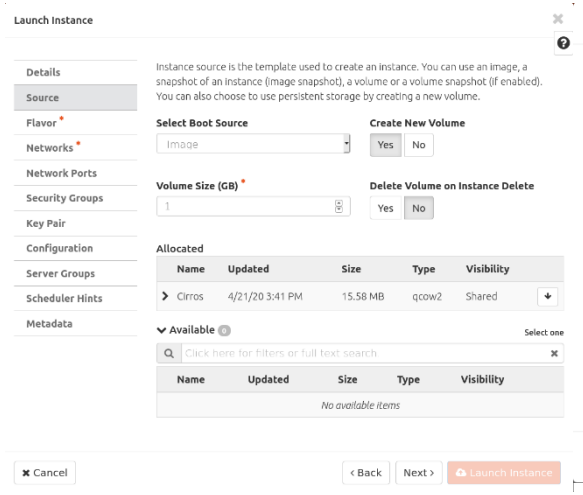
- VMs are created from images, template of a virtual hard drive in which the OS is preinstalled
- At least one image must be imported
- Download a template in your PC using the following link:
  - http://download.cirros-cloud.net/0.4.0/cirros-0.4.0-x86_64-disk.img
- Admin->Images->Create Image
  - Select the downloaded image

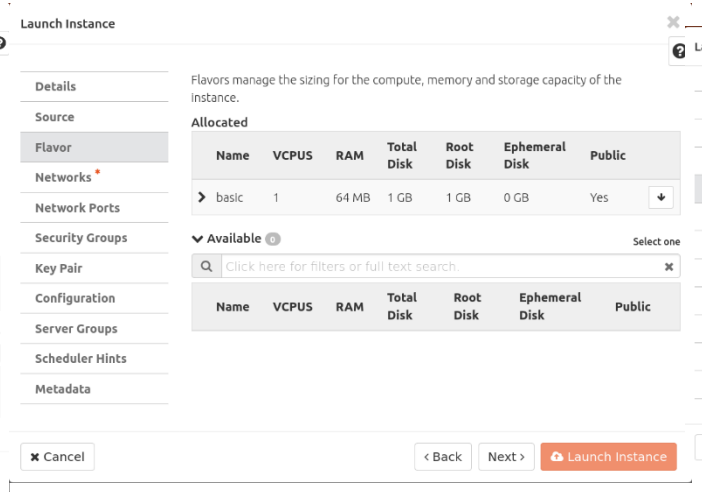


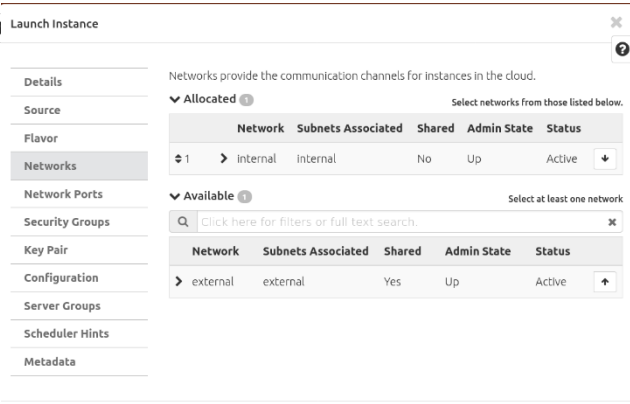
Instantiate the first VM

- Project->Compute->Instances->Launch Instance

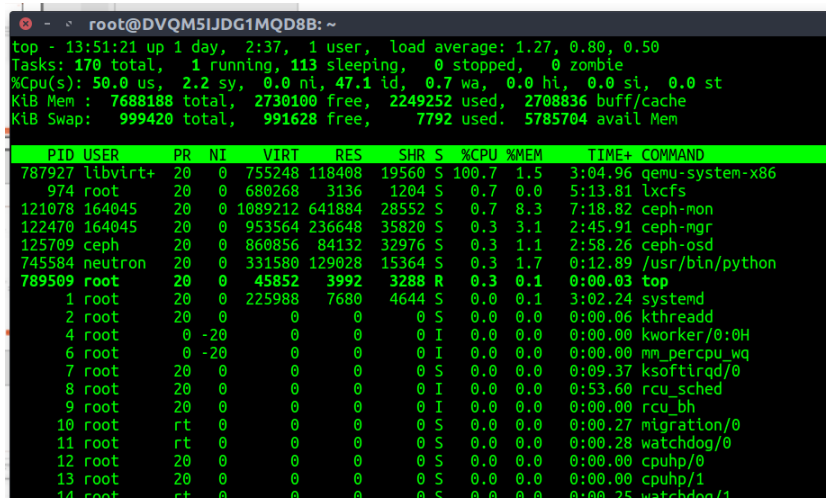








Check the image running



- Admin->Compute->Instances
- The image runs on a certain host and has a certain IP in the internal network

Instances

Project Name = | Filter | Delete Instances

Displaying 1 item

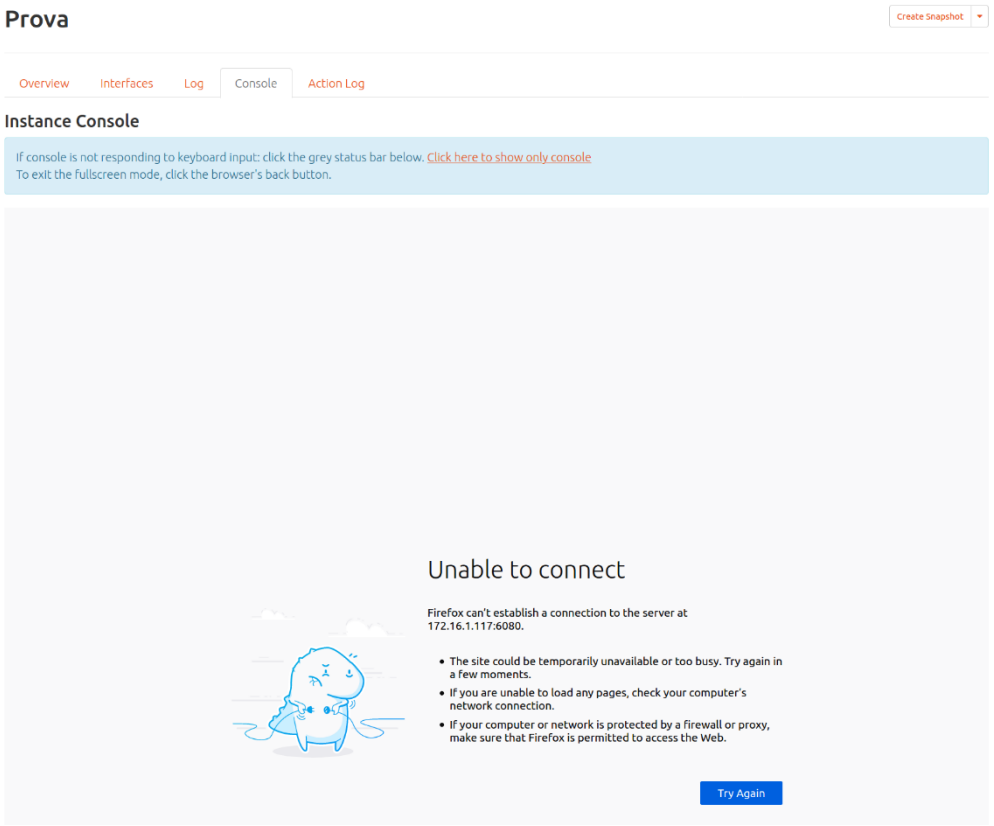
Project	Host	Name	Image Name	IP Address	Flavor	Status	Task	Power State	Age	Actions
admin	DVQM5IJDG1MQD8B	Prova	Cirros	10.1.1.56	basic	Active	None	Running	2 minutes	Rescue Instance

Displaying 1 item

Connect to the VM

- Project -> Compute -> Instances
- Select the instance and then the Console tab

VNC interface is provided by the nova controller, we need to configure a port forwarding to forward requests from the IP address of the web interface to the 6080 port of the nova controller



Forward VNC requests

- Retrieve the internal IP address of the container that hosts the nova controller
- Run juju status (**<u>on the manager</u>**) and retrieve the IP address of nova-cloud-controller

```
neutron-gateway/0*        active    idle   0          172.16.2.117                          Unit is ready
nova-cloud-controller/0*  active    idle   0/lxd/4    252.2.117.160   8774/tcp,8775/tcp     Unit is ready
nova-compute/0*           active    idle   1          172.16.2.151                          Unit is ready
  neutron-openvswitch/1   active    idle              172.16.2.151                          Unit is ready
nova-compute/1            active    idle   2          172.16.2.152                          Unit is ready
```

- Install the following rules **<u>on the controller</u>**

```
iptables -t nat -A PREROUTING -i eth0 -p tcp -m tcp --dport 6080 -j DNAT --to-destination 252.2.117.160:6080
iptables -t nat -A POSTROUTING -d 252.2.117.160/32 -o eth0 -j MASQUERADE
```

If the configuration is wrong…

- A module can be reconfigured as follows:

```
juju config --file dashboard.yaml openstack-dashboard
```

Connect to the VM

```
$
login as 'cirros' user. default password: 'gocubsgo'. use 'sudo' for root.
dsdfs login:
login as 'cirros' user. default password: 'gocubsgo'. use 'sudo' for root.
dsdfs login:
login as 'cirros' user. default password: 'gocubsgo'. use 'sudo' for root.
dsdfs login:
login as 'cirros' user. default password: 'gocubsgo'. use 'sudo' for root.
dsdfs login: _
```

Connected (unencrypted) to: QEMU (instance-0000000b)

```
^K64 bytes from 10.1.1.1: seq=1 ttl=64 time=1.271 ms
^C
--- 10.1.1.1 ping statistics ---
2 packets transmitted, 2 packets received, 0% packet loss
round-trip min/avg/max = 1.271/9.001/16.731 ms
$ ifconfig
eth0      Link encap:Ethernet  HWaddr FA:16:3E:40:EB:66
          inet addr:10.1.1.70  Bcast:10.1.1.255  Mask:255.255.255.0
          inet6 addr: fe80::f816:3eff:fe40:eb66/64 Scope:Link
          UP BROADCAST RUNNING MULTICAST  MTU:1458  Metric:1
          RX packets:88 errors:0 dropped:0 overruns:0 frame:0
          TX packets:119 errors:0 dropped:0 overruns:0 carrier:0
          collisions:0 txqueuelen:1000
          RX bytes:9086 (8.8 KiB)  TX bytes:10524 (10.2 KiB)

lo        Link encap:Local Loopback
          inet addr:127.0.0.1  Mask:255.0.0.0
          inet6 addr: ::1/128 Scope:Host
          UP LOOPBACK RUNNING  MTU:65536  Metric:1
          RX packets:0 errors:0 dropped:0 overruns:0 frame:0
          TX packets:0 errors:0 dropped:0 overruns:0 carrier:0
          collisions:0 txqueuelen:1
          RX bytes:0 (0.0 B)  TX bytes:0 (0.0 B)

$
```

```
          inet addr:127.0.0.1  Mask:255.0.0.0
          inet6 addr: ::1/128 Scope:Host
          UP LOOPBACK RUNNING  MTU:65536  Metric:1
          RX packets:0 errors:0 dropped:0 overruns:0 frame:0
          TX packets:0 errors:0 dropped:0 overruns:0 carrier:0
          collisions:0 txqueuelen:1
          RX bytes:0 (0.0 B)  TX bytes:0 (0.0 B)

$ ping 10.1.1.1
PING 10.1.1.1 (10.1.1.1): 56 data bytes
64 bytes from 10.1.1.1: seq=0 ttl=64 time=1.723 ms
64 bytes from 10.1.1.1: seq=1 ttl=64 time=0.837 ms
64 bytes from 10.1.1.1: seq=2 ttl=64 time=0.719 ms
^K64 bytes from 10.1.1.1: seq=3 ttl=64 time=0.697 ms
64 bytes from 10.1.1.1: seq=4 ttl=64 time=0.719 ms
^C
--- 10.1.1.1 ping statistics ---
5 packets transmitted, 5 packets received, 0% packet loss
round-trip min/avg/max = 0.697/0.939/1.723 ms
$ ping 8.8.8.8
PING 8.8.8.8 (8.8.8.8): 56 data bytes
^C
--- 8.8.8.8 ping statistics ---
2 packets transmitted, 0 packets received, 100% packet loss
$ _
```

Connect the VM to the internet

- VMs cannot connect to the internet, the physical gateway with address 192.168.1.1 that reroutes traffic from/to VMs to/from the internet does not exist on vlan2 network (the virtual network that we created to mimic a missing real public network)
- Let us configure the network node (the controller in our deployment) to be that node
- Configure the bridge br-ex (the bridge that collects all the traffic towards external networks) on the controller node with 192.168.1.1 address:

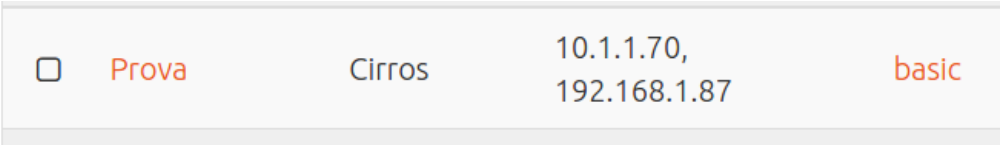
```
ethernets:
    br-ex:
        addresses:
        - 192.168.1.1/24
```

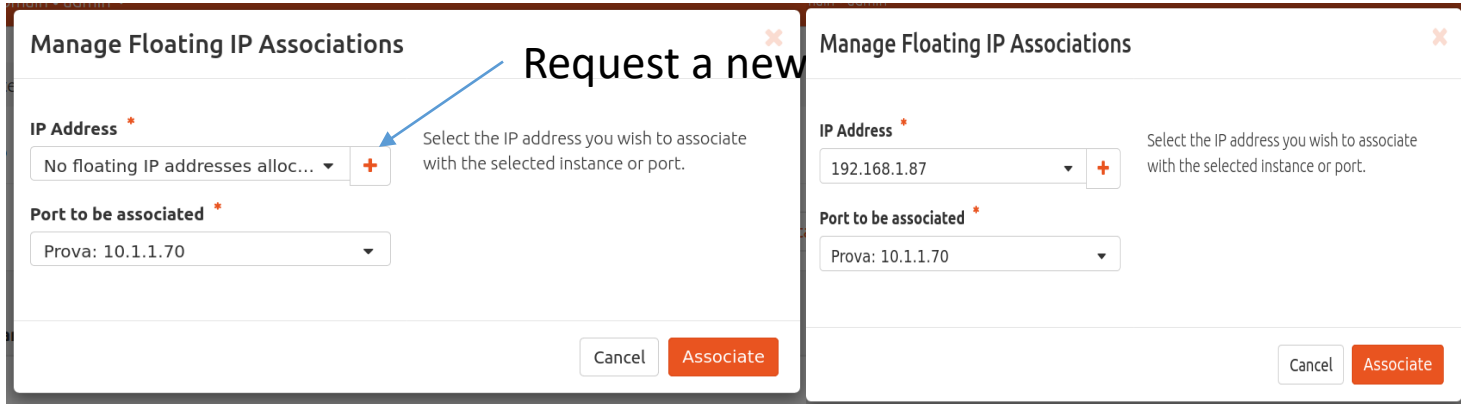
```
root@J6Y5KZ0OWNX7C42:~# ifconfig br-ex
br-ex: flags=4163<UP,BROADCAST,RUNNING,MULTICAST>  mtu 1500
        inet 192.168.1.1  netmask 255.255.255.0  broadcast 192.168.1.255
        inet6 fe80::21d:d8ff:feb7:257b  prefixlen 64  scopeid 0x20<link>
        ether 00:1d:d8:b7:25:7b  txqueuelen 1000  (Ethernet)
        RX packets 65  bytes 3086 (3.0 KB)
        RX errors 0  dropped 215  overruns 0  frame 0
        TX packets 44  bytes 3496 (3.4 KB)
        TX errors 0  dropped 0 overruns 0  carrier 0  collisions 0
```

Apply the configuration to /etc/netplan/01-netcfg.yaml and run **`netplan apply`**

Floating IPs

- VMs can be assigned a public IP address to connect to the internet via Floating IP addresses
- A Floating Ip is an IP address that is assigned to a VM to access an external network
- The IP address is not actually assigned to the VM, instead, NAT is performed by the router
- To assign a floating IP address go to the menu of the instance and select ‘Associate Floating IP’





Test connectivity towards the gateway

- Run ping 192.168.1.1 on the VM

```
$
$
$
$
$ ping 192.168.1.1
PING 192.168.1.1 (192.168.1.1): 56 data bytes
64 bytes from 192.168.1.1: seq=0 ttl=63 time=3.501 ms
64 bytes from 192.168.1.1: seq=1 ttl=63 time=0.923 ms
64 bytes from 192.168.1.1: seq=2 ttl=63 time=0.708 ms
64 bytes from 192.168.1.1: seq=3 ttl=63 time=0.704 ms
^K64 bytes from 192.168.1.1: seq=4 ttl=63 time=0.885 ms
^C
--- 192.168.1.1 ping statistics ---
5 packets transmitted, 5 packets received, 0% packet loss
round-trip min/avg/max = 0.704/1.344/3.501 ms
$
```

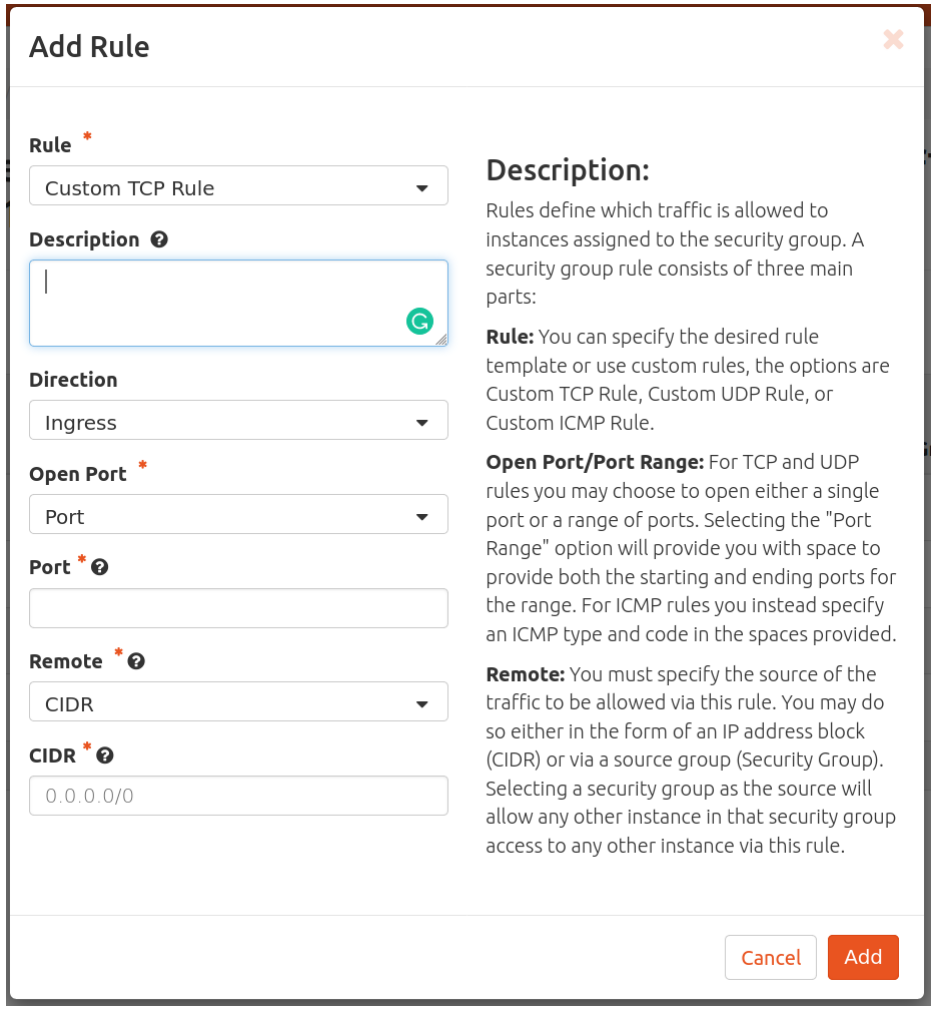
Configure the gateway

- The next step is to configure the controller to forward the traffic from br-ex to eth0 and to perform NAT to allow all the VMs on the platform to send and receive data via the controller
- IPTABLES configuration

```
iptables -t nat -A POSTROUTING -o eth0 -j MASQUERADE
```

Allow ingress traffic

- By default ingress traffic to VMs is disabled
- To enable some type of traffic:
  - Network->Security Groups->
    Manage Rules->Add Rule



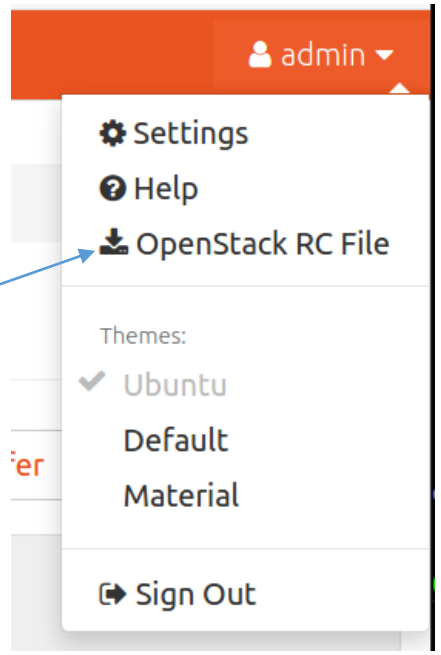
OpenStack SDK

- OpenStack provides an SDK to create applications that interact with the platform, one SDK version is available for python3
- To install it (e.g. on the controller node):

```
apt-get install python3-pip
pip3 install openstacksdk
```

- Download the authentication file and authenticate
- Upload the file on the controller and activate

```
source openstack.sh
```



Simple Application

```
import openstack

# Connect
conn=openstack.connect()
# list images
for image in conn.compute.images():
        print(image)
# list VMs
 print("List Servers:")
for server in conn.compute.servers():
       print(server)
```

Command line

- OpenStack offers a command line interface alternative to the web one

```
snap install --classic openstackclients
```

```
wget https://cloud-images.ubuntu.com/bionic/current/bionic-server-cloudimg-amd64.img
```

```
source admin.sh
```

```
openstack image create --public --disk-format qcow2 --container-format bare --file bionic-server-cloudimg-amd64.img Ubuntu
```

Other examples

- https://github.com/openstack/openstacksdk/blob/master/examples/