Openstack Guide to get started

General Openstack terminologies

a. Nova – Nova is a tool for managing Virtual Machines. It uses the underlying system's(hypervisor such as KVM, Vmware etc.). It can be accessed via Horizon by CLI or APIs
b. Swift – Swift is used for object storage in Openstack. It is horizontally scalable and is ideal for storing unstructured data such as backups, videos, pictures, user data etc.
c. Cinder – Cinder is also a block storage application in openstack but it is used to store structured data or persistent storage volumes for the VMs. Its like a hard drive for the VMs and the storage for VMs can be added by adding cinder volumes.
d. Neutron – Neutron is the networking section of openstack. All networking configurations like Virtual networks, subnets, routing etc. are provisioned via Neutron. Its like the SDN controller in Openstack
e. Glance – Where Nova was a tool to provision VMs, Glance is a tool that actually stores and manages those VMs. Templating of VMs and creating snapshots of VMs is something that Glance does.
f. Keystone – Keystone is Openstack's authentication component. It generates tokens to clients for identity validation. This is for overall infrastructure security
g. Horizon – Horizon is the front end GUI application of Openstack. The User will land at horizon and horizon will abstract all settings for the user using APIs

1. Difference between Users and Roles?

   Users: Users are the different end users that are operating on open stack. Each user may have different levels of permissions and allowed access.

   Roles: Roles are like user profiles that can be created per project. Users can be then added to be a part of these roles.

2. What is a hypervisor and which hypervisors are supported in OpenStack?

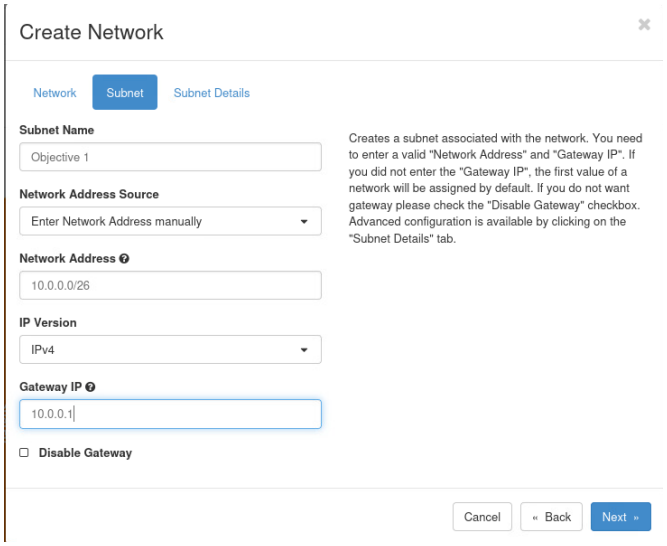
   A hypervisor is a software that runs on a bare metal server and manages resources such as compute, storage and memory. It also manages VMs and management of resources among them. OpenStack supports multiple hypervisors such as Hyper-V, Vmware Vsphere, Xen, etc. But it supposedly works best with KVM which is again open source.

3. Explain the meaning of ‘flavor’ in OpenStack.

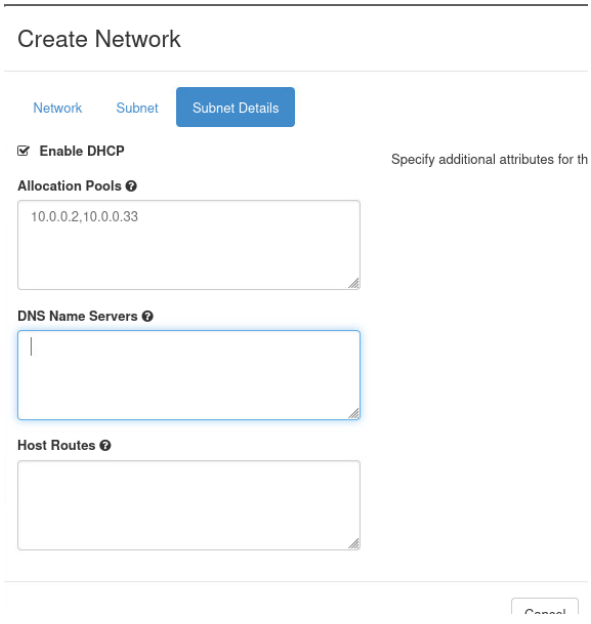
   Flavors are like templates for VMs in openstack. We can have different flavors with different levels of resources depending on the functionality of the VM. We can then create with VMs with a certain flavor to quickly spin up rather than configuring each VM manually.

Creating a new Network

1. Go to the Network topology Tab on openstack.
2. Click on Create new network. Give a Network Name
3. Then go to the subnet tab and give subnet name, network address and gateway IP



4. Go to the subnet details tab and then Enable DHCP, and allocate the pool:



5. Now click Create. In the Networks tab you will now see the new network created

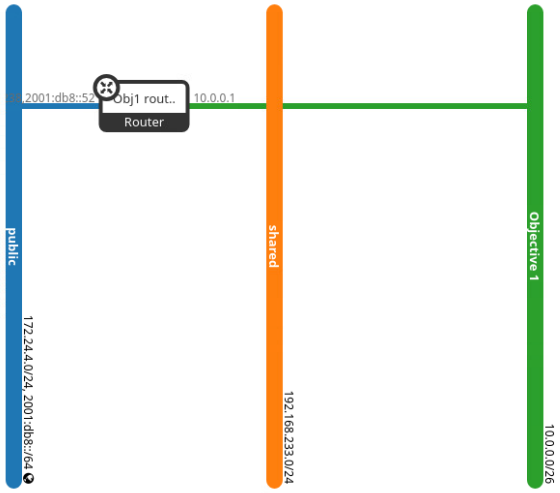
Creating and configuring a logical Router

1. Under the Network tab click on Routers.

Click on Create Router

2. Give a router name, then select the external network that you want to connect it to and then enable SNAT and then create router. This will automatically connect your router to the external network
3. In order to connect your router to your new internal network, go to the routers tab>>select your router>>go to the interfaces tab>>Add interface>>then assign an IP address to the interface and then select the internal network from the dropdown.

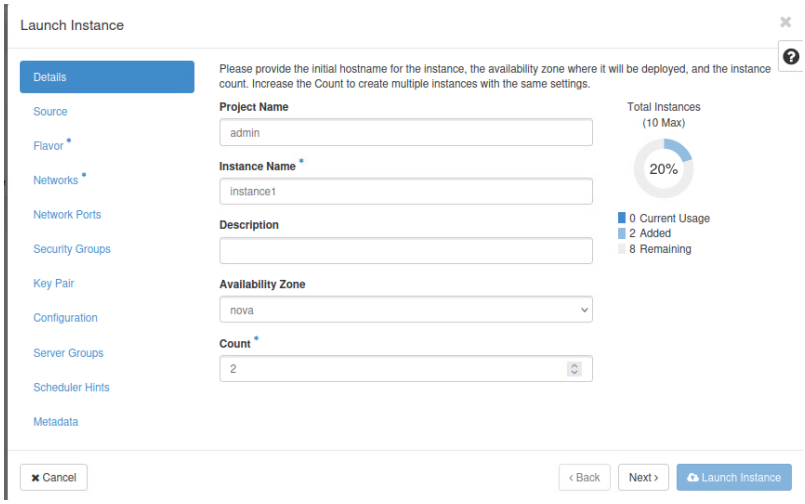
4. In the Network Topology tab, you will now see the public and the objective 1 networks connected to each other with the router:

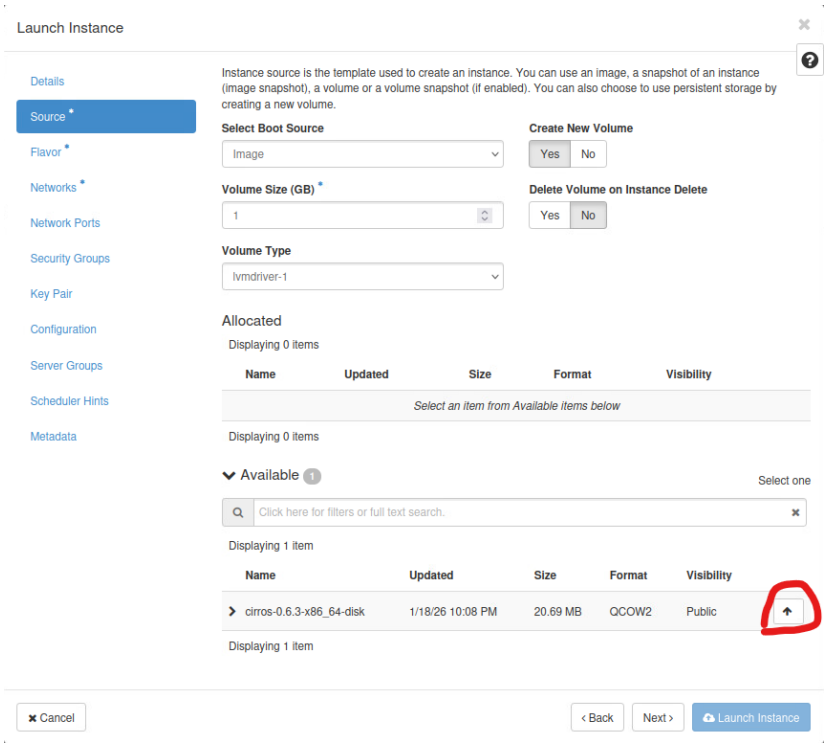


Spinning up new instances and configuring them to network

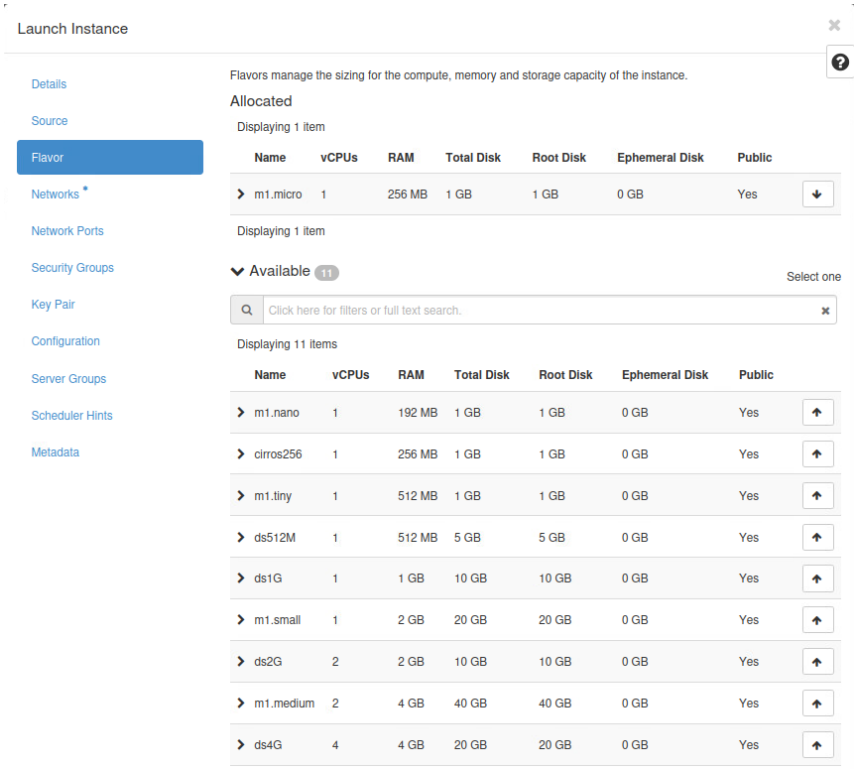
1. Under the compute tab, click on images and check if the Cirros image is present or not.
2. Now click on the instances tab and then click on Launch Instance

   Fill out the fields:

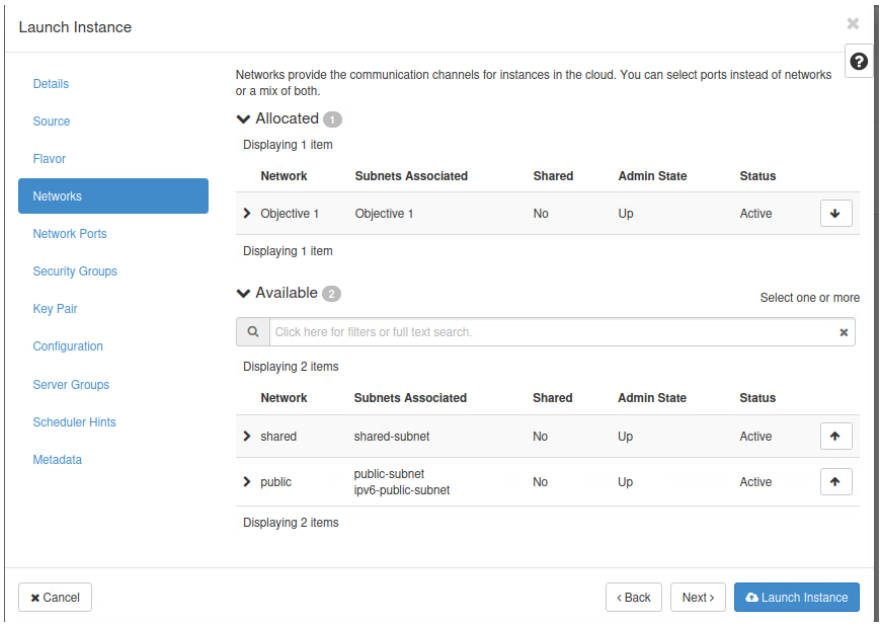




Then choose any one flavor

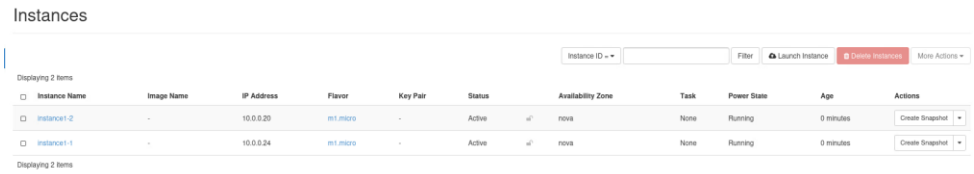


Select a network



Now click on Launch Instance

You will then see the instances being created

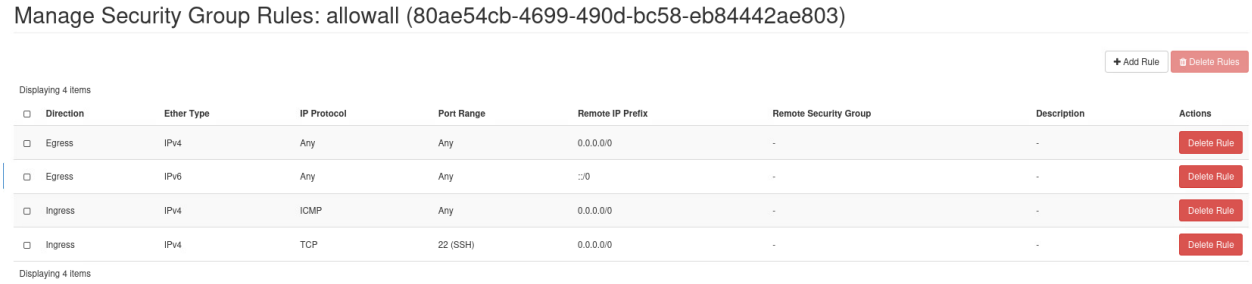


We can also see that the Ip addresses have been automatically assigned for each instance because we had enabled DHCP

Making your host VM talk to openstack instances.

Now that we have Private IP addresses associated with the instances and the router. We need to do the following.

1. Create a new security group and add the following rules for the instances and the router to be able to ping and allow ssh communication.



Assign this security group for all instances and the router interfaces.

2. Floating Ips
   Go to the instances section and for each instance, create a floating IP which would be reachable from our local machine

Lets Consider a Scenario:

You are working in a cloud firm that has a single instance of an application running on OpenStack cloud platform. The firm is planning to add a functionality to the

single running instance of the application that can autoscale/replicate itself to multiple instances whenever the compute capacity (eg. CPU cycles or memory) reaches a pre-defined threshold. Since you are familiar with the Python programming and REST API, you are being assigned a following task:

a. Write a simple Python application that can ssh into the available "cirros" instance that was created in the above objective and extract the CPU utilization information. [As an alternative, you may use ceilometer service for retrieving this telemetry data]
b. If the CPU utilization or memory usage exceeds a threshold value, for example 20%, spin up additional instances of cirros. The creation of cirros instances should be triggered whenever the usage of CPU or memory exceeds a predefined threshold. Select CPU or memory usage to your interest to define your condition to trigger the creation of additional instances. In order to collect the utilization data, you'll have to monitor its usage using appropriate commands.
c. The Python application can use Nova REST API to create additional "Cirros" instances whenever the above condition occurs.
d. The auto scaling of the instances should be handled considering following requirements:

   Max scaling size: 4 (this value denotes the maximum number of instances that should be spun)

   Increment size: 1 (this value denotes the number of instances that should be spun whenever CPU utilization exceeds threshold)

   Evaluation period: 40 (this value denotes the time period in seconds for monitoring CPU usage)

2. You can use the Linux stress tool to raise the CPU utilization of an instance above the threshold.

   Implementation:

I have created two different python scripts to each this objective: 1. Cpu_utlization.py and 2. Instance_autoscaling.py

Cpu_utilization is a simple python script that uses paramiko to ssh into the remote instance and return a cpu utilization value.

The instance_autoscaling.py is another python script that imports the get_cpu_usage function from cpu_utilization.py and then uses that to check realtime cpu usage and spin up instances if needed.

```
def get_token():
    url = "http://10.224.76.78/identity/v3/auth/tokens"
    payload = {
        "auth": {
            "identity": {
                "methods": ["password"],
                "password": {
                    "user": {
                        "name": "admin",
                        "domain": {"id": "default"},
                        "password": "pranav"
                    }
                }
            },
            "scope": {
                "project": {
                    "name": "admin",
                    "domain": {"id": "default"}
                }
            }
        }
    }
```

The instance_autoscaling.py has 3 functions and a loop logic

Functions:

1. Keystone authentication: It uses the identity API to create authentication tokens for the user and for the project

2. Creating instances using Nova API: We need to populate a few server data such as name, imageref, flavorref and network ID that we want to spin up the new server

```python
def create_instance(token):
    nova_url = "http://10.224.76.78/compute/v2.1/servers"

    server_data = {
        "server": {
            "name": "autoscale-vm",
            "imageRef": "2caa938d-243a-42e8-a454-6e8b894af38a",
            "flavorRef": "84",
            "networks": [{"uuid": "38cf01ad-82d7-477c-b115-b5a8231c7354"}]
        }
    }

    headers = {
        "X-Auth-Token": token,
        "Content-Type": "application/json"
    }

    r = requests.post(nova_url, json=server_data, headers=headers)
    print("Nova API response:", r.json())
```

3. Get_instance_count() function: This basically polls how many instances are currently spun up for this project. It again uses the nova API to achieve the same

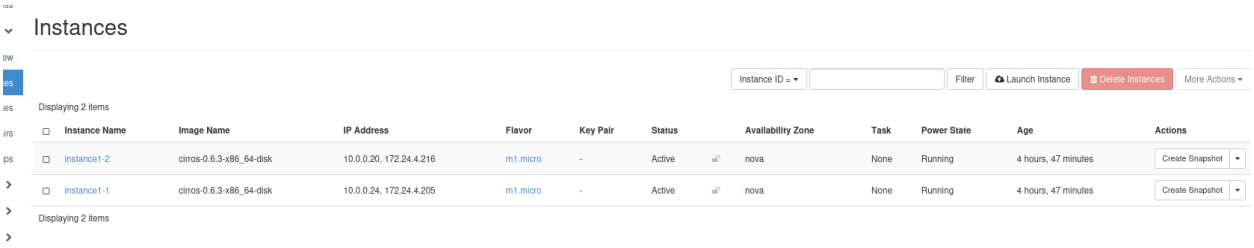
```python
def get_instance_count(token):
    url = "http://10.224.76.78/compute/v2.1/servers/detail"
    headers = {"X-Auth-Token": token}

    r = requests.get(url, headers=headers)
    data = r.json()

    # Count ACTIVE instances only
    active = [s for s in data["servers"] if s["status"] == "ACTIVE"]
    return len(active)
```

Working of autoscaling

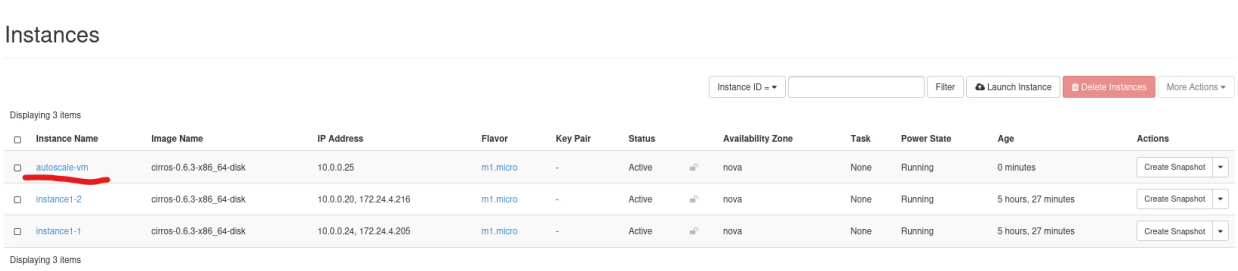
Instances before




```
student@csci5380-vm1-prpr4002:~/nvo_lab$ /bin/python3 /home/student/nvo_lab/instance_autoscaling.py
CPU: 0.00%
CPU normal
CPU: 0.00%
CPU normal
```

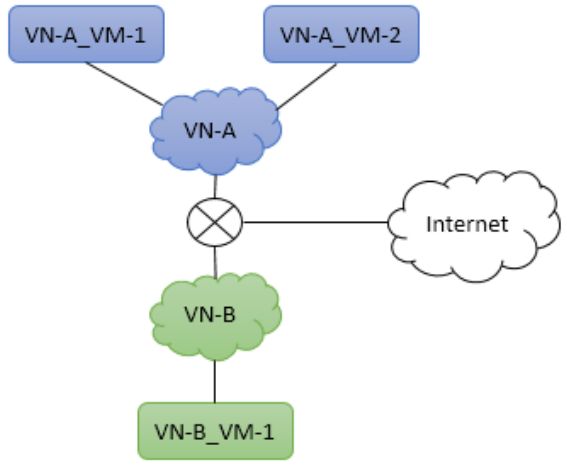
After increasing CPU usage on the current running instance

```
student@csci5380-vm1-prpr4002:~/nvo_lab$ /bin/python3 /home/student/nvo lab/instance autoscaling.py
CPU normal
CPU: 0.00%
CPU normal
CPU: 100.00%
High utilization detected
Current instance count: 2
Scaling out by 1 instance...
Nova API response: {'server': {'id': '9dcbd025-2f61-42a2-ab3c-6ac7fce6b481', 'links': [{'rel': 'self', 'href': 'http://10.224.76.78
/compute/v2.1/servers/9dcbd025-2f61-42a2-ab3c-6ac7fce6b481'}, {'rel': 'bookmark', 'href': 'http://10.224.76.78/compute/servers/9dcb
d025-2f61-42a2-ab3c-6ac7fce6b481'}], 'OS-DCF:diskConfig': 'MANUAL', 'security_groups': [{'name': 'default'}], 'adminPass': 'QXRaLa8
4Q3x3'}}
```



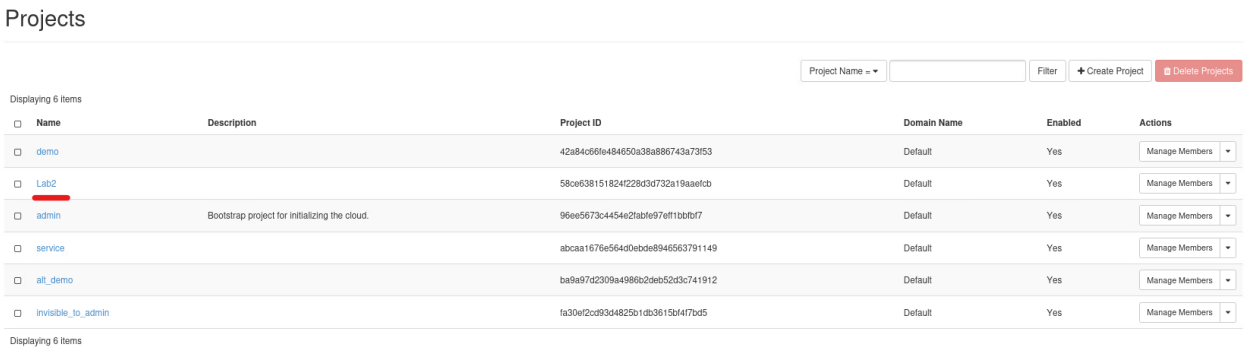
Multi-tenants

Name	Address
VN-A	192.168.100.0/24
VN-B	192.168.200.0/24
VN-A_VM-1	192.168.100.2
VN-A_VM-2	192.168.100.3
VN-B_VM-1	192.168.200.2



Creating project, user, flavor and image

1. Within OpenStack UI Identity tag, create a project called lab2. Then create a user called lab2_admin and attach it to the project lab2.



2. Within OpenStack UI Admin tag, create a VM Flavor called **ngn.tiny** with the following setting (or the setting that works for your VM image):

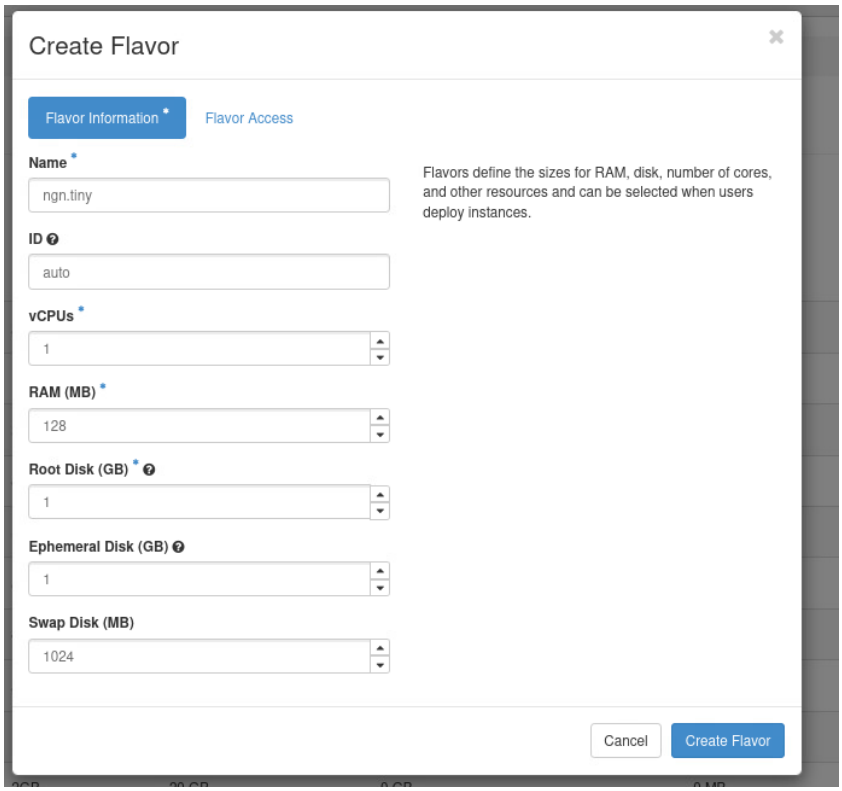
   vCPU = 1

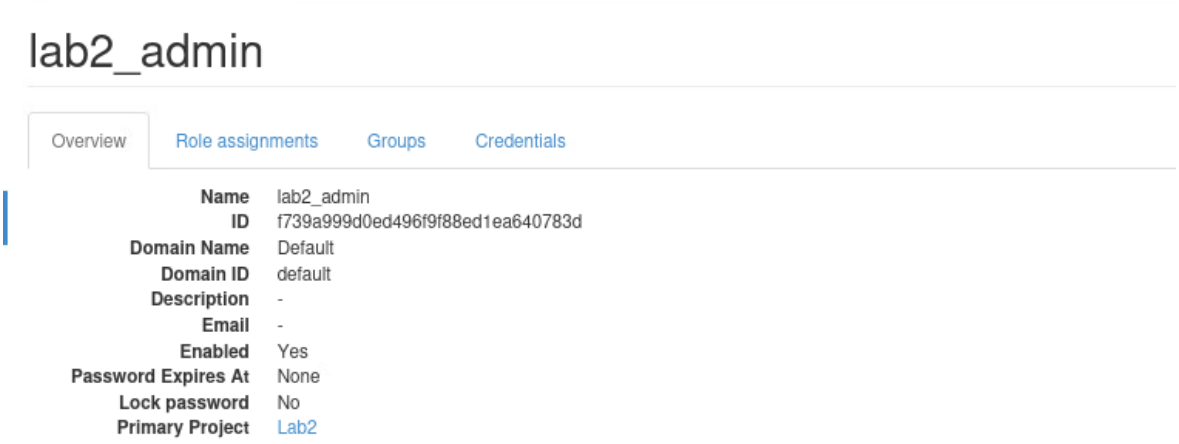
   RAM = 128MB

   Root Disk = 1GB

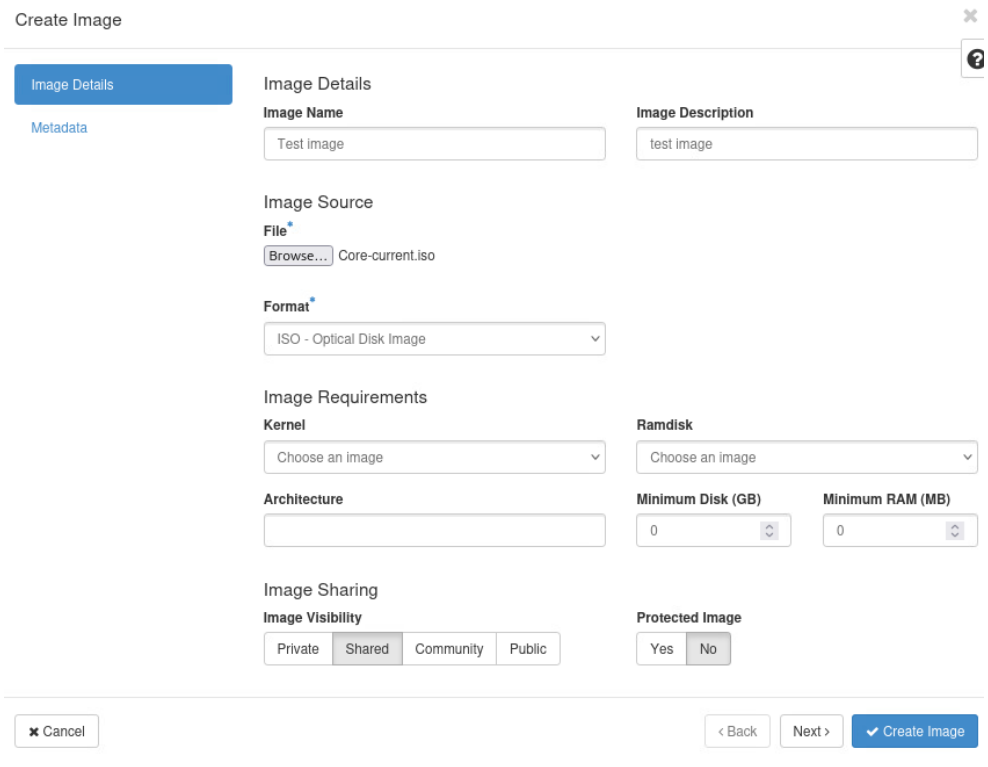
Ephemeral Disk = 1GB

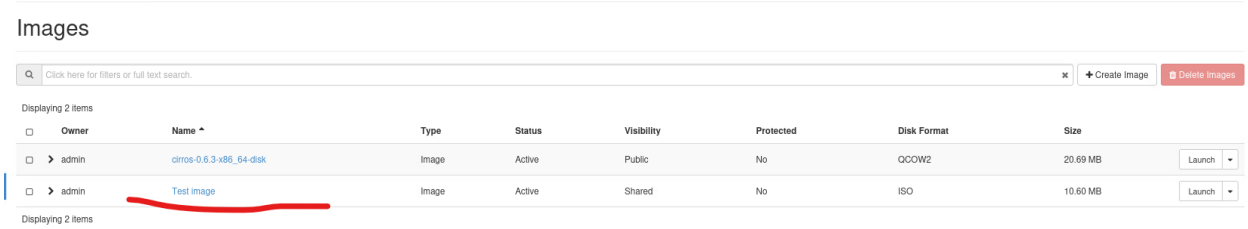
Swap Disk = 1GB





3. Within OpenStack UI Admin tag, upload a VM image into OpenStack. You can use this URL: http://tinycorelinux.net/7.x/x86/release/Core-current.iso or https://docs.openstack.org/image-guide/obtain-images.html.
   Remember to make it public.





4. Before proceeding, logout and login with your newly created user lab2_admin.

Setup Virtual Networks

1. Login back into OpenStack UI, within the Project tag, create a new Network called VN-A with network address 192.168.100.0/24.
2. Repeat the above steps to create a second network VN-B with network address 192.168.200.0/24.

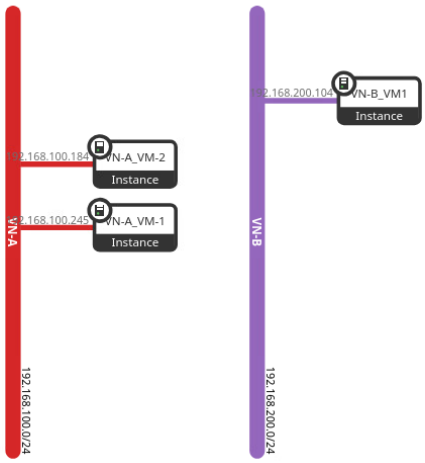
Networks

Name = | Filter | + Create Network | Delete Networks

Displaying 5 items

Name	Subnets Associated	Shared	External	Status	Admin State	Availability Zones	Actions
Objective 1	**Objective 1** 10.0.0.0/26	Yes	No	Active	UP	-	Edit Network
shared	**shared-subnet** 192.168.233.0/24	Yes	No	Active	UP	-	Edit Network
VN-A	**VN-A** 192.168.100.0/24	Yes	No	Active	UP	-	Edit Network
VN-B	**VN-B** 192.168.200.0/24	Yes	No	Active	UP	-	Edit Network
public	**public-subnet** 172.24.4.0/24 **ipv6-public-subnet** 2001:db8::/64	No	Yes	Active	UP	-	Edit Network

Displaying 5 items



Section 3: Launch VM instances

Launch the following VMs using the flavor and image created in Section 1.

1. Launch VN-A_VM-1 and VN-A_VM-2 into virtual network VN-A.
2. Launch VN-B_VM-1 into virtual network VN-B.

Instances

Instance ID = | Filter | Launch Instance | Delete Instances | More Actions

Displaying 3 items

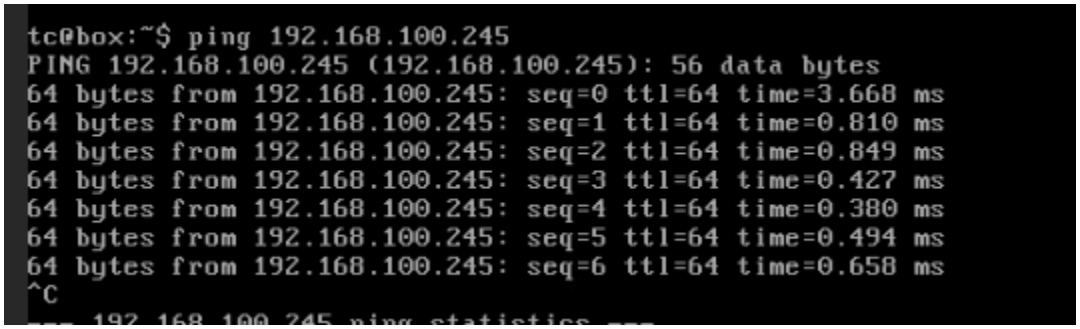
Instance Name	Image Name	IP Address	Flavor	Key Pair	Status	Availability Zone	Task	Power State	Age	Actions
VN-B_VM1	-	192.168.200.104	ngn.tiny	-	Build	nova	Networking	No State	0 minutes	Associate Floating IP
VN-A_VM-2	Test image	192.168.100.184	ngn.tiny	-	Active	nova	None	Running	0 minutes	Create Snapshot
VN-A_VM-1	Test image	192.168.100.245	ngn.tiny	-	Active	nova	None	Running	2 minutes	Create Snapshot

Displaying 3 items

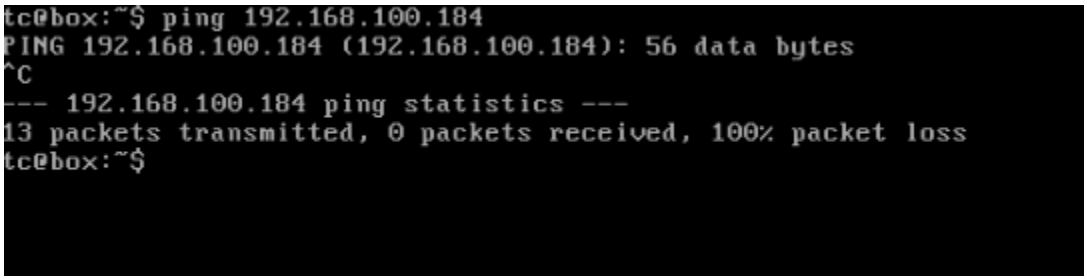
Section 4: Ping testing

1. Use the console within OpenStack UI to test if VMs in VN-A can ping each other, while the VM in VN-B cannot reach VMs in VN-A.

VN-A_VM-1 to VN-A_VM-2 ping

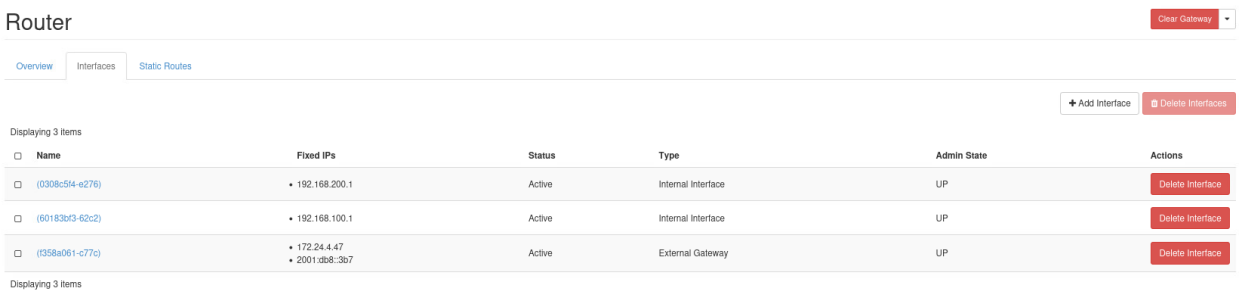


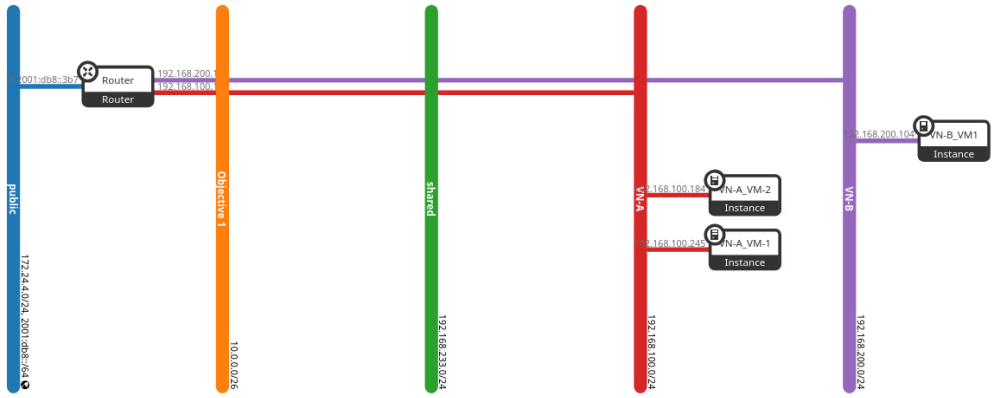
VN-B_VM-1 to VN-A_VM1 ping fail



2. Assign floating IP's to the VM's both in VN-A and VN-B, and test connectivity to the Internet.

To create floating IPs, we need to create a router which connects both these networks and then associate floating Ips. We will also need to create rules to permit ICMP and SSH





Manage Security Group Rules: default (f32976aa-d5fe-47ba-9cce-b2b745209b74)

+ Add Rule | Delete Rules

Displaying 7 items

Direction	Ether Type	IP Protocol	Port Range	Remote IP Prefix	Remote Security Group	Description	Actions
Egress	IPv4	Any	Any	0.0.0.0/0	-	-	Delete Rule
Egress	IPv4	ICMP	Any	0.0.0.0/0	-	-	Delete Rule
Egress	IPv6	Any	Any	::/0	-	-	Delete Rule
Ingress	IPv4	Any	Any	-	default	-	Delete Rule
Ingress	IPv4	ICMP	Any	0.0.0.0/0	-	-	Delete Rule
Ingress	IPv4	TCP	22 (SSH)	0.0.0.0/0	-	-	Delete Rule
Ingress	IPv6	Any	Any	-	default	-	Delete Rule

Displaying 7 items

Floating IP addresses:

Displaying 3 items

Instance Name	Image Name	IP Address	Flavor	Key Pair	Status	Availability Zone	Task	Power State	Age	Actions
VN-B_VM1	Test image	192.168.200.104, 172.24.4.173	ngn.tiny	-	Active	nova	None	Running	34 minutes	Create Snapshot
VN-A_VM-2	Test image	192.168.100.184, 172.24.4.50	ngn.tiny	-	Active	nova	None	Running	35 minutes	Create Snapshot
VN-A_VM-1	Test image	192.168.100.245, 172.24.4.198	ngn.tiny	-	Active	nova	None	Running	36 minutes	Create Snapshot

Displaying 3 items

Ping test to google.com

```
tc@box:~$ ping 142.250.72.206
PING 142.250.72.206 (142.250.72.206): 56 data bytes
64 bytes from 142.250.72.206: seq=0 ttl=111 time=8.612 ms
64 bytes from 142.250.72.206: seq=1 ttl=111 time=4.670 ms
64 bytes from 142.250.72.206: seq=2 ttl=111 time=5.030 ms
64 bytes from 142.250.72.206: seq=3 ttl=111 time=6.739 ms
64 bytes from 142.250.72.206: seq=4 ttl=111 time=4.757 ms
64 bytes from 142.250.72.206: seq=5 ttl=111 time=2.381 ms
64 bytes from 142.250.72.206: seq=6 ttl=111 time=3.457 ms
64 bytes from 142.250.72.206: seq=7 ttl=111 time=4.810 ms
^C
--- 142.250.72.206 ping statistics ---
8 packets transmitted, 8 packets received, 0% packet loss
round-trip min/avg/max = 2.381/5.057/8.612 ms
tc@box:~$ S
```

Network policies in OpenStack

Now manage the network policy inside OpenStack, such that:

1. VN-A_VM-1 can ping VN-B_VM-1, but VN-A_VM-2 cannot ping VN-B_VM-1.
2. VN-B_VM-1 can go out to the Internet, but VN-A_VM-1 and VN-A_VM-2 cannot.

Manage Security Group Rules: default (f32976aa-d5fe-47ba-9cce-b2b745209b74)

+ Add Rule | Delete Rules

Displaying 11 items

Direction	Ether Type	IP Protocol	Port Range	Remote IP Prefix	Remote Security Group	Description	Actions
Egress	IPv4	Any	Any	0.0.0.0/0	-	-	Delete Rule
Egress	IPv4	ICMP	Any	192.168.100.245/32	-	-	Delete Rule
Egress	IPv4	ICMP	Any	192.168.200.0/24	-	-	Delete Rule
Egress	IPv4	TCP	Any	192.168.200.0/24	-	-	Delete Rule
Egress	IPv6	Any	Any	::/0	-	-	Delete Rule
Ingress	IPv4	Any	Any	-	default	-	Delete Rule
Ingress	IPv4	ICMP	Any	192.168.200.0/24	-	-	Delete Rule
Ingress	IPv4	ICMP	Any	192.168.100.245/32	-	-	Delete Rule
Ingress	IPv4	TCP	Any	192.168.200.0/24	-	-	Delete Rule
Ingress	IPv4	TCP	22 (SSH)	0.0.0.0/0	-	-	Delete Rule
Ingress	IPv6	Any	Any	-	default	-	Delete Rule

Displaying 11 items

I have created ICMP and TCP rules in order to satisfy the above two requirements