Tetra project OpenCloudEdge

Short tutorial OpenStack

ANDREA PELLICCIA, LUCA GATTOBIGIO

28/11/2019

Abstract

The main purpose of this presentation is to show how Openstack's core services work. It uses easy-to-understand examples, such as creating virtual networks and subnets, launching instances and attaching volumes to them, and using telemetry tools for autoscaling resources.

Outline

1. Openstack configuration
2. Creating virtual networks via GUI (Horizon)
3. Launch an instance
4. Attach volume
5. Associate a floating IP
6. Autoscaling simulation

1. Openstack configuration:

CONTROLLER Node

KEYSTONE	*Identity service*
HORIZON	*Dashboard*
NOVA	*Compute conductor, API*
GLANCE	*Image service*
NEUTRON	*Networking*
CINDER	*Block storage scheduler*
CEILOMETER	*Telemetry*
AODH	*Alarm*
HEAT	*Orchestration*

COMPUTE Node

NOVA	*Compute*
NEUTRON	*Networking agents*
CEILOMETER	*Telemetry agents*

STORAGE Node

CINDER	*Block storage compute*

2. Creating virtual networks via GUI (Horizon, Neutron, Keystone)

The only core service required by the *Horizon* dashboard is the *Keystone* Identity service. The *Horizon* dashboard can be used in combination with other services, such as the Image, Compute, and Networking services.

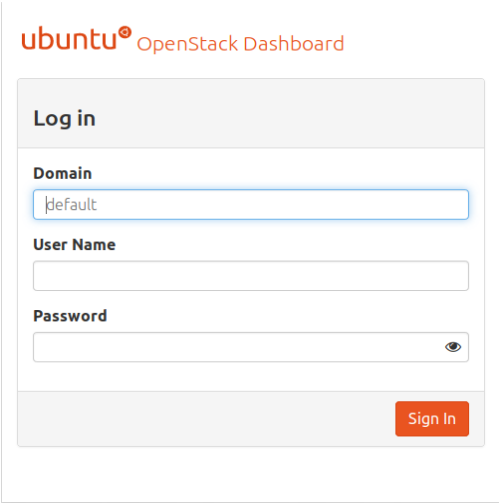

Figure 2.1 Horizon login page

The *Horizon* dashboard can be used to manage services like the *Neutron* networking service. *Horizon* allows creating virtual networks and subnets in *Neutron* by setting the name of the reference project, the network type, and the IP subnet ranges. This among others allows creating private networks used for internal communication between the compute instances of

a certain project or user. It also allows specifying the external, physical network referred to as the *provider network*. Upon creating a virtual network, *Neutron* will provision a Dynamic Host Configuration Protocol (DHCP) server to distribute IP addresses within the configured subnet ranges inside said virtual network.

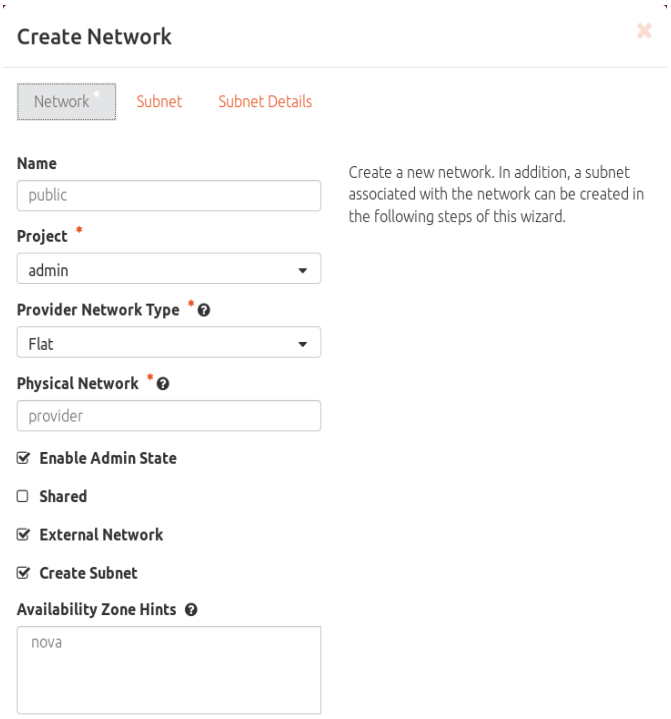

Figure 2.2 Create Network

Interconnection between virtual and physical networks, such as for example between two virtual private networks and the physical *provider* network, is provided by virtual routers. These virtual routers can perform tasks like IPv4 Source/Destination Network Address Translation (SNAT/DNAT). Routers require a (virtual) network interface on each of the networks to be connected.

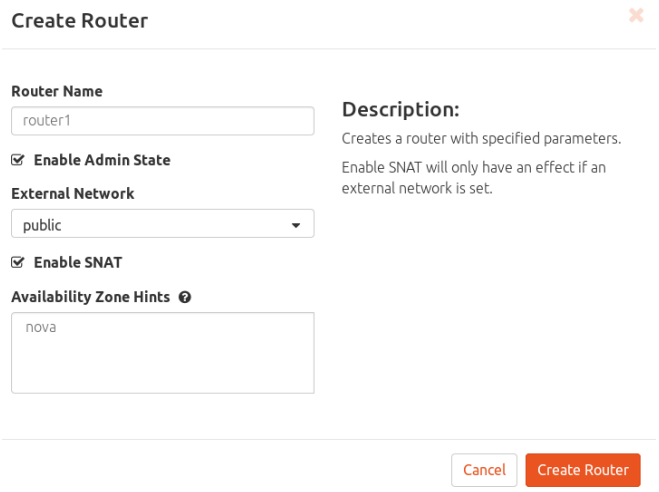

Figure 2.3 Create router

The network topology presented in Figure 2.4 indicates the interconnection between an external *provider* network named “public” and the virtual networks “private” and “private2” via the *Neutron* router named “router1”. The corresponding IP addresses or the router interfaces and subnets of all three networks are visible.

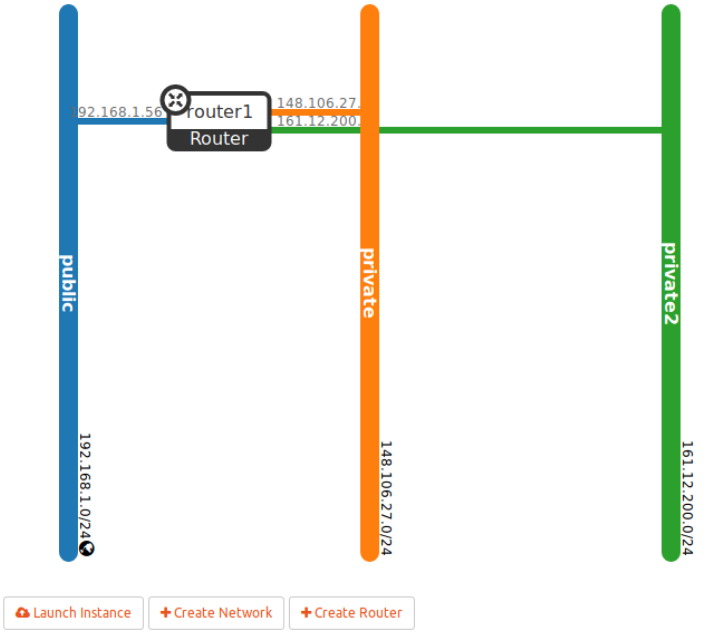

Figure 2.4 Network topology

3. Launching an instance (Horizon, Neutron, Nova, Glance, Keystone)

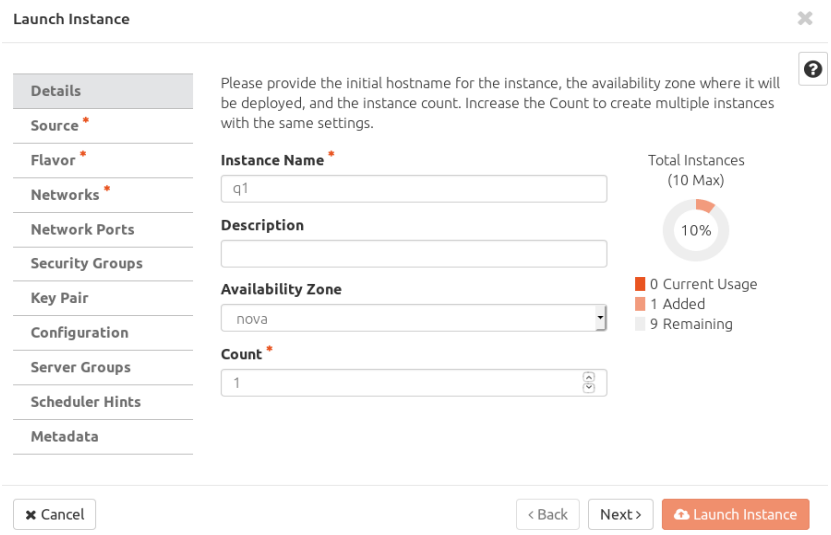

Figure 3.1 Launching an instance

The *Horizon* dashboard allows the creation of compute instances via a stepwise approach. This includes the required specification of the instance name and quantity (Figure 3.1), *Glance* image or volume to be deployed (Figure 3.2), virtual machine *flavor* to specify vCPU and RAM configuration *(Figure 3.3)*, and network selection (Figure 3.4). It also allows setting optional configuration fields such as security groups which are used as firewall rules, etc.

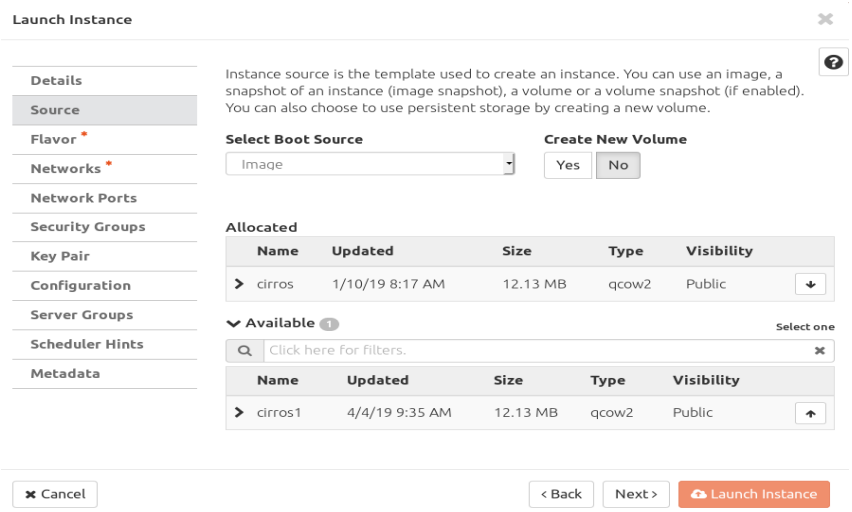

Figure 3.2 Launching an instance – image selection

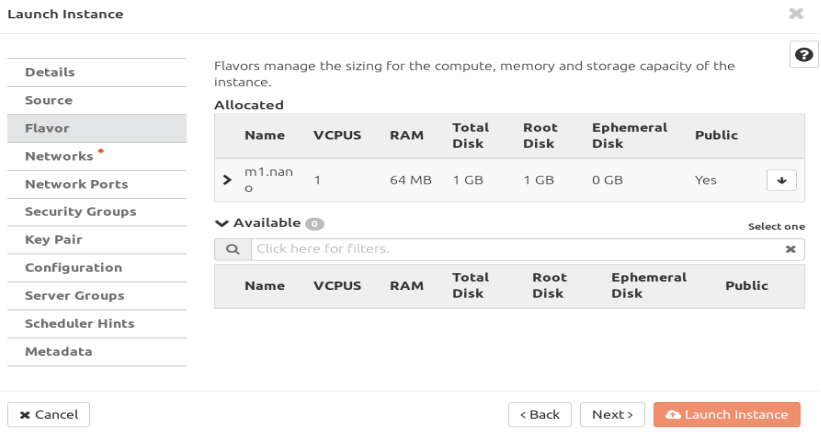

Figure 3.3 Launching an instance – flavor selection

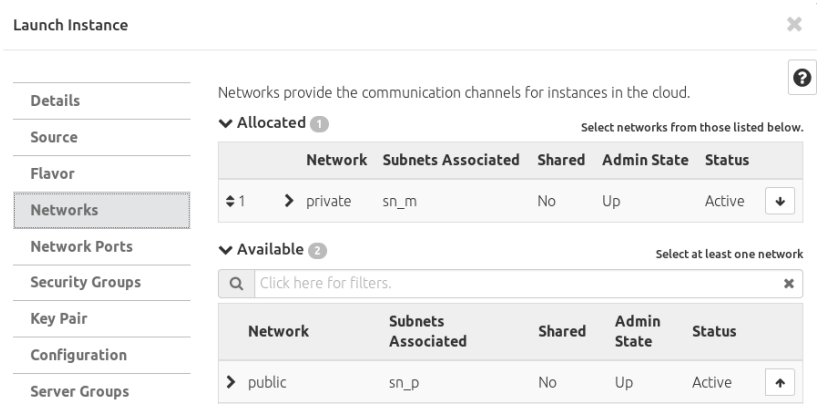

Figure 3.4 Launching an instance – network selection

After creating and launching the image, it is possible to open a remote console with the instance from within the *Horizon* dashboard.

4. Attach persistent storage volumes (Horizon, Nova, Cinder, Keystone)

The *Glance* image used for deploying instances provides non-persistent storage. Persistent data storage can be provided by attaching a *Cinder* block storage volume on which the data can be stored. The *Horizon* dashboard indicates which volumes are attached to which instances.

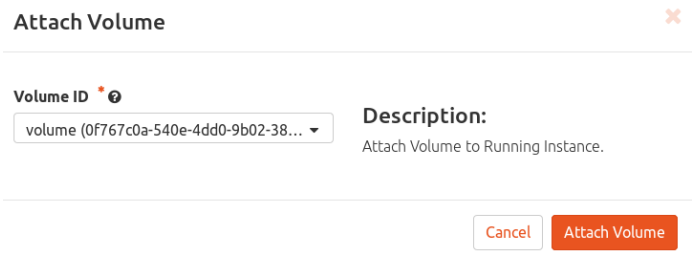

Figure 4.1 Attaching a storage volume to an instance

5. Assigning floating IP addresses (Horizon, Nova, Neutron, Keystone)

Floating IP addresses enable reaching OpenStack instances connected to a virtual private network from the external network, using IPv4. *Neutron*'s virtual routers provide Destination Network Address Translation (DNAT) between the assigned floating IP address and the instance's IP address on the virtual private network.

Floating IP addresses can be associated and dissociated at any time. The example provided in Figure 5.1 shows the assignment of floating IP *192.168.1.51* to IP *148.106.27.58* in a private subnet.

Note that for most server applications the floating IP (192.168.1.51) should be an Internet routable IPv4 address. However, this is not possible at time of testing with our current physical network setup but will be re-evaluated once our server infrastructure is in place.

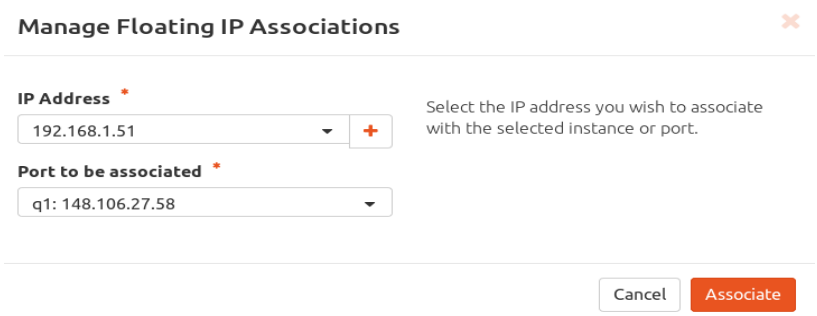

Figure 5.1 Floating IP associations

6. Autoscaling test (Horizon, Heat, Ceilometer, Aodh, Nova, Glance, Neutron, Keystone)

In this section we provide an autoscaling example using the OpenStack *Heat* orchestration service. The example shows OpenStack's ability to deal with certain situations by performing an action on *alarm* triggers. In this case, it shows the capability to dynamically scale resources based on the workload of a fictitious server.

The following resources are created through the configuration file *autoscaling-group.yml*:

- Private virtual network with subnet 10.0.20.0/24 in which the instances will be created.

- Router for combining the created private network with the cloud infrastructure's *provider* network.

- Security group for specifying firewall rules.

- **Autoscaling group**. This specifies the resources to be scaled with another separate template, which passes the necessary parameters and metadata to be used to find the metrics via *Ceilometer* and *Aodh*, as well as the minimum and maximum number of instances to be created.
  A "nested stack" has been used to simplify the template. By doing so, one can easily define the set of resources to be scaled separately by providing a separate template.

- **Two alarms** which monitor the metric *cpu_util* of all resources containing the *"server_group"* metadata, as specified by the autoscaling group above. These alarms alert the respective resources to which the *scale-up* and *scale-down* task is assigned.

- **Two Heat resources** of the *ScalingPolicy* type, which have the task of scaling the resources based on the respective alarms associated with them.

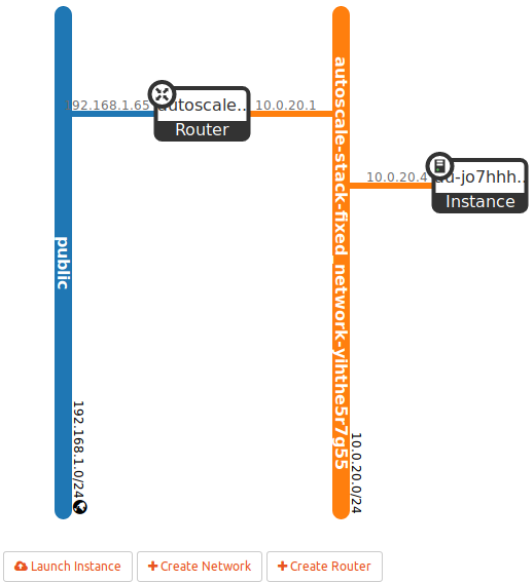

Figure 6.1 Stack topology

Figure 6.1 shows the initial situation in which one server instance is operational.

```
studente@fondamenti:~$ openstack alarm list
+--------------------------------------+---------------------------------------+---------------------------------------+-------+----------+---------+
| alarm_id                             | type                                  | name                                  | state | severity | enabled |
+--------------------------------------+---------------------------------------+---------------------------------------+-------+----------+---------+
| a5c2f00b-f404-41b0-bd21-1ad42201ca0a | gnocchi_aggregation_by_resources_threshold | autoscale-stack-cpu_alarm_high-6ylimol56sxy | ok    | low      | True    |
| fb3ed649-07c6-4d57-b406-13b0920e4a7e | gnocchi_aggregation_by_resources_threshold | autoscale-stack-cpu_alarm_low-77wqbhmbe6h3 | alarm | low      | True    |
+--------------------------------------+---------------------------------------+---------------------------------------+-------+----------+---------+
```

Figure 6.2 Alarm list

After the initial data collection phase, it is possible to check the configured alarms, as is depicted in Figure 6.2.

By manually causing a high load on the compute instance for a period of time, one can verify that the “autocale-stack-cpu_alarm_high” alarm is triggered and that the *Heat* orchestrator launches an additional instance to increase the available resources. This process is depicted in Figure 6.3.

This process continues until the maximum number of configured instances is reached, or until the CPU usage of these instances drops below a set threshold.

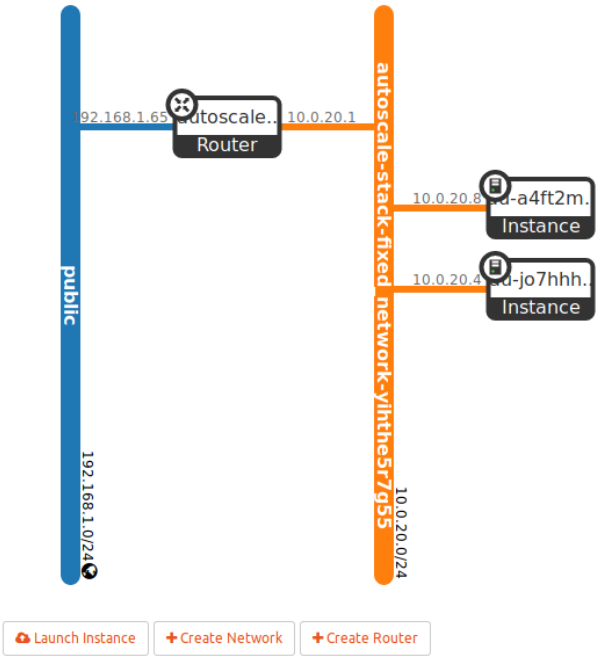

Figure 6.3 Automatic scale-up

When the CPU load of these instances drops below a minimum threshold, one can observe how some instances are removed to release resources that are no longer needed.