

1 Introduction

This document has the main purpose to present a basic template to be fed to Openstack Heat to create a sample 'service' i.e. a composition of interconnected VMs. In the follow a reference template is described and the created topology is displayed via the resulting Openstack GUI snapshots. (In this installation, Openstack Neutron is running directly over OVS, so no SDN controller is used).

2 Template

The following template has been created using the HOT (Heat Orchestration Template) format.

The following template create two hosts (Host1 and Host2), each one on a different subnet (5.0.0.0/24 and 6.0.0.0/24) and a router with one interface per subnet. The hosts are not given external connectivity. All network resources are instantiated ex-novo via the template.

heat_template_version: 2013-05-23

description: >

This template creates 2 hosts. Each Host is connected to a private network. A router is created with one interface per subnet to allow host connectivity.

resources:

test_network_1:
 type: OS::Neutron::Net
 properties:
 name: test_network_1

test_subnet_1:
 type: OS::Neutron::Subnet
 properties:
 network_id: { get_resource: test_network_1 }
 name: test_subnet_1
 cidr: 5.0.0.0/24
 gateway_ip: 5.0.0.1
 allocation_pools:
 - start: 5.0.0.100
 end: 5.0.0.200

host1:
 type: OS::Nova::Server
 properties:
 name: host1
 image: cirros-0.3.1-x86_64-uec
 flavor: m1.nano
 networks:
 - port: { get_resource: host1_port }

host1_port:
 type: OS::Neutron::Port
 properties:
 network_id: { get_resource: test_network_1 }
 fixed_ips:
 - subnet_id: { get_resource: test_subnet_1 }

test_network_2:
 type: OS::Neutron::Net
 properties:
 name: test_network_2

```

test_subnet_2:
  type: OS::Neutron::Subnet
  properties:
    network_id: { get_resource: test_network_2 }
    name: test_subnet_2
    cidr: 6.0.0.0/24
    gateway_ip: 6.0.0.1
    allocation_pools:
      - start: 6.0.0.100
        end: 6.0.0.200
host2:
  type: OS::Nova::Server
  properties:
    name: host2
    image: cirros-0.3.1-x86_64-uec
    flavor: m1.nano
    networks:
      - port: { get_resource: host2_port }
host2_port:
  type: OS::Neutron::Port
  properties:
    network_id: { get_resource: test_network_2 }
    fixed_ips:
      - subnet_id: { get_resource: test_subnet_2 }
router:
  type: OS::Neutron::Router
router_interface1:
  type: OS::Neutron::RouterInterface
  properties:
    router_id: { get_resource: router }
    subnet_id: { get_resource: test_subnet_1 }
router_interface2:
  type: OS::Neutron::RouterInterface
  properties:
    router_id: { get_resource: router }
    subnet_id: { get_resource: test_subnet_2 }

```

This template, once fed to Openstack Heat project allocates proper resources and create the desired topology as shown in the following snapshots taken from Openstack GUI (Horizon).

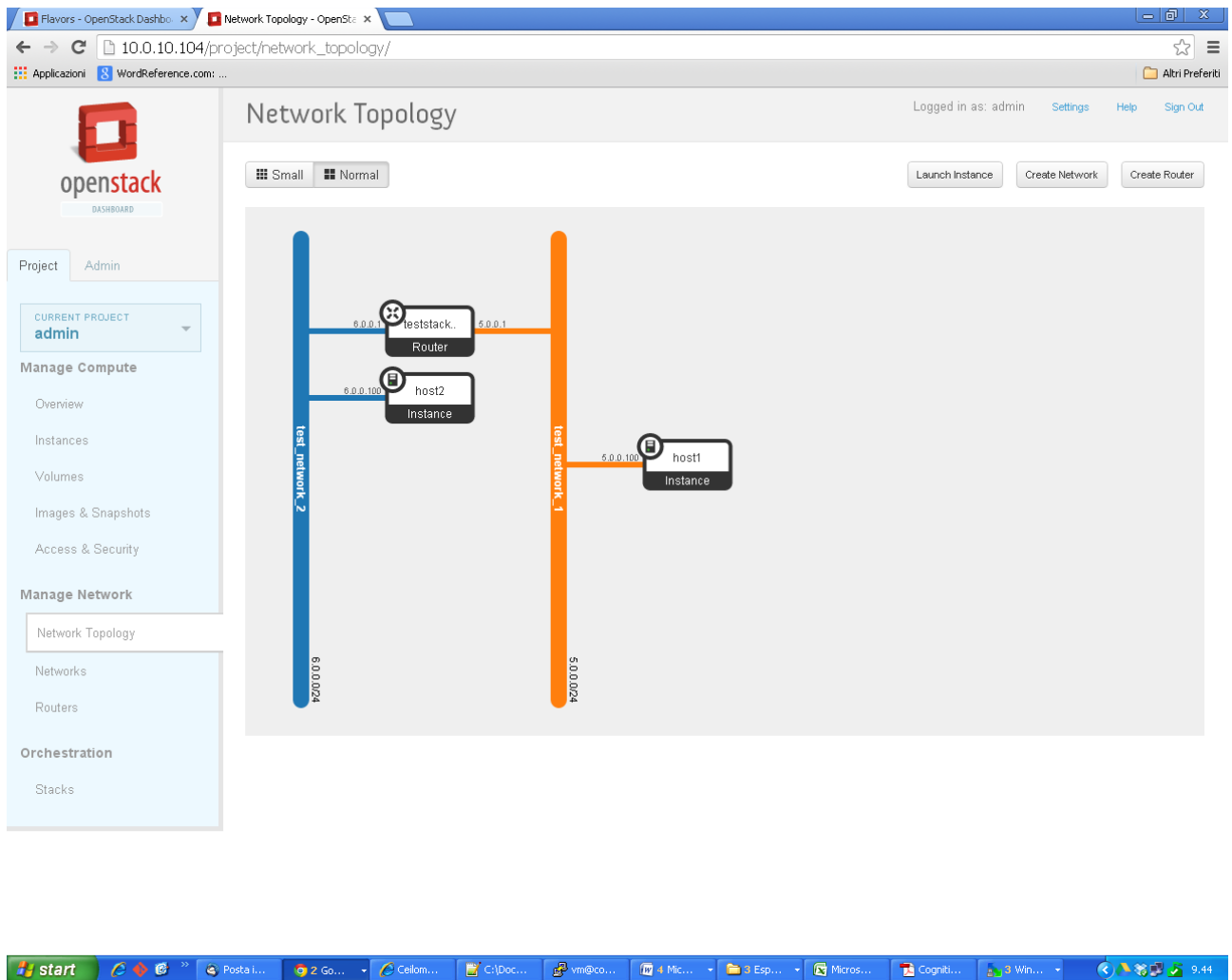
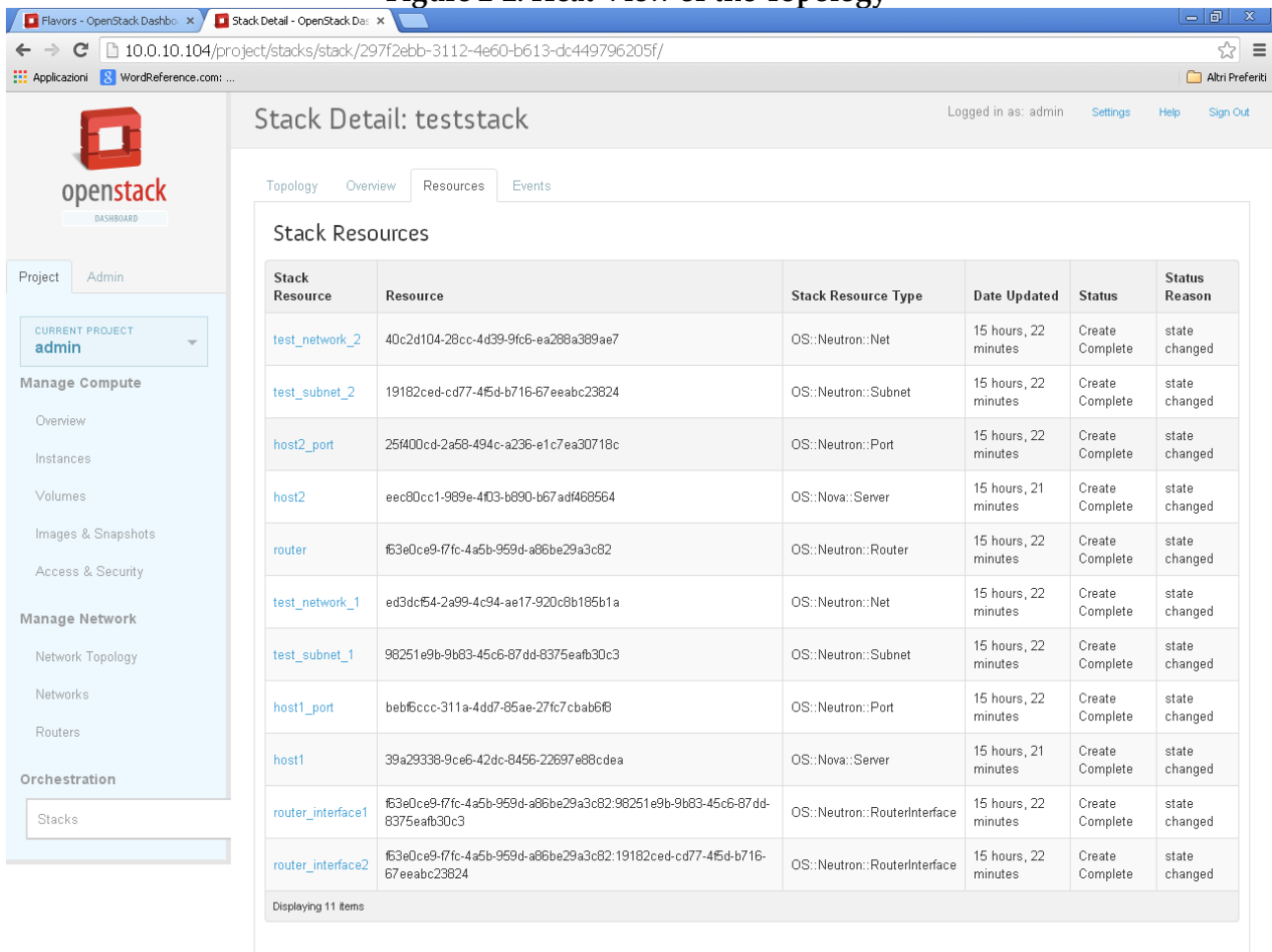


Figure 2-1: Topology created

Router interface

Figure 2-2: Heat View of the Topology



Stack Detail: teststack

Topology Overview Resources Events

Stack Resources

Stack Resource	Resource	Stack Resource Type	Date Updated	Status	Status Reason
test_network_2	40c2d104-28cc-4d39-9fc6-ea28a389ae7	OS::Neutron::Net	15 hours, 22 minutes	Create Complete	state changed
test_subnet_2	19182ced-cd77-4f5d-b716-67eeabc23824	OS::Neutron::Subnet	15 hours, 22 minutes	Create Complete	state changed
host2_port	251400cd-2a68-494c-a236-e1c7ea30718c	OS::Neutron::Port	15 hours, 22 minutes	Create Complete	state changed
host2	eec80cc1-989e-4f03-b890-b67adf468564	OS::Nova::Server	15 hours, 21 minutes	Create Complete	state changed
router	f63e0ce9-f7fc-4a5b-959d-a86be29a3c82	OS::Neutron::Router	15 hours, 22 minutes	Create Complete	state changed
test_network_1	ed3dcf54-2a99-4c94-ae17-920c8b185b1a	OS::Neutron::Net	15 hours, 22 minutes	Create Complete	state changed
test_subnet_1	98251e9b-9b83-45c6-87dd-8375eafb30c3	OS::Neutron::Subnet	15 hours, 22 minutes	Create Complete	state changed
host1_port	beb6ccc-311a-4dd7-85ae-27fc7cbab6f8	OS::Neutron::Port	15 hours, 22 minutes	Create Complete	state changed
host1	39a29338-9ce6-42dc-8456-22697e88cdea	OS::Nova::Server	15 hours, 21 minutes	Create Complete	state changed
router_interface1	f63e0ce9-f7fc-4a5b-959d-a86be29a3c82:98251e9b-9b83-45c6-87dd-8375eafb30c3	OS::Neutron::RouterInterface	15 hours, 22 minutes	Create Complete	state changed
router_interface2	f63e0ce9-f7fc-4a5b-959d-a86be29a3c82:19182ced-cd77-4f5d-b716-67eeabc23824	OS::Neutron::RouterInterface	15 hours, 22 minutes	Create Complete	state changed

Displaying 11 items

Figure 2-3: Openstack resources allocated by Heat via the template.

3 Quality of Service in Heat template

In the latest official release of OpenStack (Havana), QoS parameters for network resources are not yet implemented and integrated. A very preliminary draft for modelling QoS within Neutron framework can be found at <https://wiki.openstack.org/wiki/Neutron/QoS>.

In this draft proposal the following assumptions are made:

- A new logical object (qos) should be created as neutron resource. This new object will contain qos parameters (e.g. policy-rate = BW in kpbs). At the moment two types of QoS objects are allowed:
 - DSCP, for marking packets with a DSCP mark
 - Ratelimit, to throttle a port or all ports attached to a network based on a bandwidth value
- This new object will be applied to a network (meaning that the traffic related to all host belonging to that network will be managed according to the selected policy) or will be applied to an interface (meaning that only the traffic on that interface will be managed according to the selected policy)

The state-of-art is still incomplete, and many point are left open, for example how to manage parameters like latency or jitter (related to the traffic among two VMs), how to deal with packet classification and marking, etc. However this modelling is a good starting point that can be extended to include further parameters and specifications.

Following the approach mentioned above, a QoS parameter can be added in the Heat template as follows:

```
heat_template_version: 2014-03-23

description: >
  This template creates 2 hosts. Each Host is connected to a private network. A router is
  created with one interface per subnet to allow host connectivity.

resources:
  test_network_1:
    type: OS::Neutron::Net
    properties:
      name: test_network_1

  qos_1:
    type: OS::Neutron::qos
    properties:
      type: rate_limit
      policy: 1024 kpbs

  test_subnet_1:
    type: OS::Neutron::Subnet
    properties:
      network_id: { get_resource: test_network_1 }
      name: test_subnet_1
      cidr: 5.0.0.0/24
      gateway_ip: 5.0.0.1
```

```

allocation_pools:
  - start: 5.0.0.100
    end: 5.0.0.200
host1:
  type: OS::Nova::Server
  properties:
    name: host1
    image: cirros-0.3.1-x86_64-uec
    flavor: m1.nano
    networks:
      - port: { get_resource: host1_port }
host1_port:
  type: OS::Neutron::Port
  properties:
    network_id: { get_resource: test_network_1 }
    fixed_ips:
      - subnet_id: { get_resource: test_subnet_1 }
      qos_id: { get_resource: qos_1 }

```

We propose to extend this approach defining a more structured QoS resource (**OS::Neutron::qos**) that identify a group of QoS parameters (**OS::Neutron::qos_param**) and the related traffic classifiers (**OS::Neutron::classifier**). The latter is an optional object that can be applied to limit the enforcement of a given QoS parameter to a selected subset of traffic.

In some cases, the classifier parameter can identify the relationship with a given port resource (i.e. a network interface): this is useful for parameters like delay or jitter that may be referred only to a specific pair of VMs' ports. However, these types of parameters can be also simply applied to a network resource: in this case they will be referred to all the possible pairs of ports attached to the network itself.

An example of QoS resource is provided in the following:

```

heat_template_version: 2014-03-25

description: >
  This template creates 2 hosts. Each Host is connected to a private network. A router is
  created with one interface per subnet to allow host connectivity.

resources:
  test_network_1:
    type: OS::Neutron::Net
    properties:
      name: test_network_1
      qos_id: { get_resource: qos_1 }

  qos_1:
    type: OS::Neutron::qos
    properties:
      qos_parameter: { get_resource: qos_p1 }
      qos_parameter: { get_resource: qos_p3 }

  qos_2:
    type: OS::Neutron::qos
    properties:
      qos_parameter: { get_resource: qos_p2 }

  qos_p1:
    type: OS::Neutron::qos_param
    properties:
      type: rate_limit

```

```

    policy: 1024 kpbs
    classifier: { get_resource: classifier_c2 }

qos_p2:
  type: OS::Neutron::qos_param
  properties:
    type: delay
    policy: 2 ms
    classifier: { get_resource: classifier_c1 }

qos_p3:
  type: OS::Neutron::qos_param
  properties:
    type: delay
    policy: 4 ms

classifier_c1:
  type: OS::Neutron::classifier
  properties:
    type: destinationIf
    policy: { get_resource: host2_port }

classifier_c2:
  type: OS::Neutron::classifier
  properties:
    type: L3_protocol
    policy: udp

test_subnet_1:
  type: OS::Neutron::Subnet
  properties:
    network_id: { get_resource: test_network_1 }
    name: test_subnet_1
    cidr: 5.0.0.0/24
    gateway_ip: 5.0.0.1
    allocation_pools:
      - start: 5.0.0.100
        end: 5.0.0.200

host1:
  type: OS::Nova::Server
  properties:
    name: host1
    image: cirros-0.3.1-x86_64-uec
    flavor: m1.nano
    networks:
      - port: { get_resource: host1_port }

host2:
  type: OS::Nova::Server
  properties:
    name: host1
    image: cirros-0.3.1-x86_64-uec
    flavor: m1.nano
    networks:
      - port: { get_resource: host2_port }

host3:
  type: OS::Nova::Server
  properties:
    name: host1
    image: cirros-0.3.1-x86_64-uec
    flavor: m1.nano
    networks:
      - port: { get_resource: host3_port }

host1_port:
  type: OS::Neutron::Port
  properties:
    network_id: { get_resource: test_network_1 }
    fixed_ips:
      - subnet_id: { get_resource: test_subnet_1 }

```

```

qos_id: { get_resource: qos_2 }

host2_port:
  type: OS::Neutron::Port
  properties:
    network_id: { get_resource: test_network_1 }
    fixed_ips:
      - subnet_id: { get_resource: test_subnet_1 }

host3_port:
  type: OS::Neutron::Port
  properties:
    network_id: { get_resource: test_network_1 }
    fixed_ips:
      - subnet_id: { get_resource: test_subnet_1 }

```

In this example we have three hosts (*host1*, *host2*, *host3*) that are connected to a single network, each of them with a network interface (*host1_port*, *host2_port*, *host3_port*). At the network level, we have defined a QoS resource *qos1*, so that the maximum delay acceptable between any port attached to the network is 4 ms (see *qos_p3* resource) and the UDP traffic is limited to 1024 kbps on every port (see *qos_p1* resource and the associated classifier *classifier_c2*).

On the other hand, since we need a more restricted value for the maximum delay between *host1* and *host2*, we have defined an additional QoS resource *qos2*, which refers to the QoS parameter resource *qos_p2*. Here the classifier *classifier_c1* specifies that the QoS requirement needs to be enforced only for the subset of traffic that is destined to network interface in *host2*. Finally, the QoS resource *qos2* is attached to the *host1_port*. The resulting topology is shown in Figure 3-4.

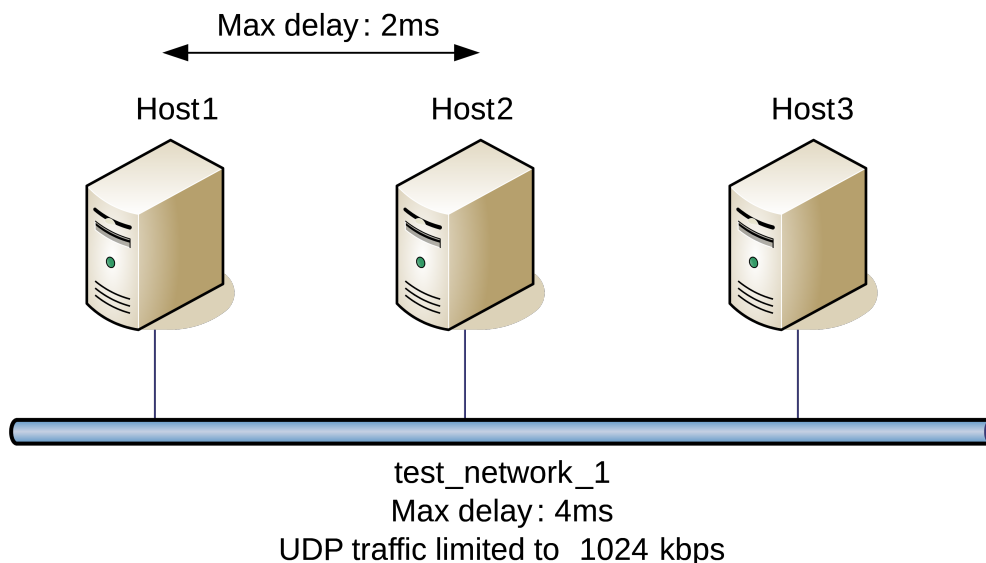


Figure 3-4: Example of network topology with QoS parameters.

The sample values included in the classifier resources of the previous example are related to a specific port resource or to the value of a field in the IP packet header (in this case the protocol field). This can be generalized to any value (or combination) of the fields in L2, L3 and L4 headers.

The following table shows the possible types of QoS parameters that can be defined.

QoS parameter type	Description	Applicable to resource	Notes
Rate-limit	Maximum bandwidth of traffic allowed when traffic shaping is applied.	Port, Network	Applicable to both routers' and hosts' ports.
DSCP	DSCP value for marking packets	Port, Network (meaningful only in combination with classifier)	Applicable to both routers' and hosts' ports.
Minimum Reserved Bandwidth	Minimum value of bandwidth that is reserved for that traffic flow.	Port, Network	
Delay	Maximum acceptable latency (one direction) between two hosts.	Port (it requires a classifier to indicate the destination port) Network	
Jitter	Maximum acceptable jitter between two hosts.	Port (it requires a classifier to indicate the destination port) Network	

Table 1 – QoS parameters

It should be noted that while rate-limit and DSCP parameters corresponds to real configuration that can be directly translated to the configuration of the physical devices, this is not valid for parameters like delay and jitter. However, their indication on the user side can lead to other types of resource allocation or configuration decisions at the orchestration level (e.g. VMs to be interconnected with low delay can be allocated in the same server, or in the same DC; low values of jitter can be achieved through proper configuration of queues and traffic priorities in the DC routers, etc.). The minimum reserved bandwidth may be supported through specific DC router configuration, or dedicated network technologies (e.g. sub-wavelength optical technologies) and, in case of inter-DC interconnections, through Bandwidth on Demand service.