OpenStack on OpenStack -- Boostrapping Open vSwitch and Quantum

Author: Florian Otel <florian.otel@gmail.com> ; @FlorianOtel ; +FlorianOtel
Date: Mar 2013

1. Introduction

This article is intended as an explanatory guide for installing and configuring Open vSwitch and OpenStack Quantum virtual network service directly from current source code base (aka "trunk").

At the time at this writing (Mar 2013) there is a rich set of OpenStack installation guides in various configurations -- see e.g. this excellent guide on how to deploy a multi-node DevStack installation for testing purposes; or this vendor-specific guide; or here for a vendor-independent set of guides for different types of configurations. However, even if the content herein will be overlapping with those resources in that it contains detailed installation instructions, the intention of this article is markedly different:

- The main focus of this post is virtualized network services in OpenStack. As such it is intended to be an explanatory guide into the functionality of Open vSwitch and Quantum and how the different components interact in providing those virtualized network services. This as opposed to a step-by-step installation guide of all OpenStack services *per se* in various topologies and underlying components -- which is the focus of the installation guides mentioned above. While for the sake of completion we will include detailed installation instructions for all OpenStack services and tools auxiliary to the focus of this post (e.g. installation and configuration of Keystone, Swift, Glance, etc), all those configuration steps are briefly documented in the appendices. For detailed description of those steps please see the installation guides mentioned above or our own series of articles on this subject.
- As the title of the article suggests we will describe how the Quantum network service and Open vSwitch (OVS) are bootstrapped directly form their respective source code repositories. However, for all other OpenStack services and auxiliary tools we will be using Linux distribution-supplied packages -- but with customized configurations to suit our needs.
- The servers that are used in this setup are virtual machines themselves running in a public cloud environment. While for this post we used hpcloud.com OpenStack-based public cloud services -- hence the title of this post -- this can be easily replicated in any virtualized environment.

For the rest of this document we will assume that underlying operating system is Ubuntu Linux 12.04 LTS. For OpenStack services we will be using the versions packaged by the Ubuntu Cloud Archive -- which at the time of this writing (Mar 2013) is the "Folsom" release of OpenStack (more exactly the 2012.2.1 Release). Unless otherwise noted all operating system commands are to be issued with "root" credentials.

2. Preparations: From isolated servers in a public cloud to multiple unfiltered L3 networks

As mentioned above the servers we used in this article were virtual machines hosted in a public cloud environment. As such they were subject to particular limitations:

- While they have both a "public IP address" (Internet routable) and a "private IP address', the servers have only one network interface (normally assigned the "private IP address" of the server).
- The traffic between those servers is subject to SG (Security Group) restrictions. In particular, the SG for the servers in our setup only allows HTTP (tcp/80) and SSH (tcp/22) incoming traffic.

However, in a typical multi-node OpenStack environment we need several, logically distinct underlying networks -- e.g. one network for management / control traffic; another distinct network for network traffic to/from VMs; a network for reaching Quantum / OpenStack external network resources; etc. As such the Security Group -based network traffic restrictions are very cumbersome to handle given the multiple server-to-server connections that occur between the different OpenStack components running on different servers.

In order to overcome these limitations we choosed to implement those underlying networks as logically distinct network overlays, with one overlay for each network type. While several solutions are possible we choosed SSH-based VPNs tunnels for implementing those overlays. In particular:

- The "management" network is an SSH tunnel mesh established between the public IP addresses of all the servers in our setup.
- The "data" network is established between the "Compute" server (the server hosting or Nova VM instances) and "Network" node (the server providing DHCP services and L3 / NAT services to VM instances).

For more details about the different server roles in our setup and the different networks that we put in place please see the diagram in the next section.

While it is not the intention of this text to provide in-depth guide on how to set up SSH-based VPNs infrastructures, here is a brief synopsis on how these networks can be implemented:

On the SSH client side (which can be whichever server in our setup) we enable VPN tunneling in "/etc/ssh/ssh_config"

```
cat <SSHEOF >>/etc/ssh/ssh_config
    Tunnel yes
    ServerAliveInterval 60

SSHEOF
```

On the corresponding SSH server side we enable VPN tunneling in "/etc/ssh/sshd_config" and then restart "sshd" daemon (the "ssh" service):

```
cat <SSHDEOF >>/etc/ssh/ssh_config
   PermitTunnel yes
```

```
SSHDEOF
#
/etc/init.d/ssh restart
```

At this point an SSH VPN tunnel between two servers can be established as follows:

From one server (be that "Folsom2") acting as SSH client we initiate, an SSH connection to other server's public IP address -- be that "Folsom1" -- by logging in as root using a pre-shared key (this SSH key can be generated and distributed in advance using standard SSH mechanisms):

```
root@Folsom2:~# ssh -fNT -w 99:99 -v -2 -i ./tunnelkey root@<the_public_ip_address_of_Folsom1> /bin/true
```

As a result on both "Folsom2" and "Folsom1" we will have a "tun99" point-to-point interface. On both ends of the tunnel we can now use this interface for routing the traffic to the private IP address of the other server through the tunnel, as follows:

On server "Folsom2" we route all traffic to "Folsom1" private IP address through the "tun99" interface:

```
root@Folsom2:~# ifconfig tun99 0.0.0.0 mtu 9000
root@Folsom2:~# route add -host <the_private_ip_address_of_Folsom1> dev tun99
```

And then perform the corresponding actions on the "Folsom 1" server:

```
root@Folsom1:~# ifconfig tun99 0.0.0.0 mtu 9000
root@Folsom1:~# route add -host <the_private_ip_address_of_Folsom2> dev tun99
```

At this point all the traffic between the private IP addresses of "Folsom1" and "Folsom2" flows through the SSH tunnel as an unfiltered point-to-point link (i.e. traffic between the private IP addresses of server "Folsom1" and "Folsom2" is not subject to their respective Security Group constraints).

Same procedure can be performed between any other two nodes in our desired setup (with the corresponding adjustment in "tunXX" tunnel interface index via ssh's "-w" option and, respectively, the corresponding private IP addresses of the two servers).

A few notes on this procedure:

- The procedure above assumes a "public IP address" and a "private IP address", with the "private IP address" being re-purposed (via adding a host route) to carry the tunnel traffic. While this is a typical scenario in public cloud environment, in e.g. a virtualized environment that role -- the "private IP address" used for the host route -- has to be substituted accordingly. This can be done by by e.g. assigning the "tunXX" interface IP addresses at both tunnel endpoints, as follows:

  ```
  root@Folsom2:~# ifconfig tun99 192.168.1.1 192.168.1.2 mtu 9000
  ```

  and, respectively, at the other end of the tunnel:

  ```
  root@Folsom1:~# ifconfig tun99 192.168.1.2 192.168.1.1 mtu 9000
  ```

  In this way we form a "numbered" point-to-point link between "Folsom1" and "Folsom2".

- As mentioned above, this mechanism can be used for instantiating several overlay networks. While the dual IP addressing scheme use in public cloud environments allow us to re-purpose the "private IP address" as a numbering scheme for one of those overlays, for any additional networks that we need to instantiate we will need to devise a different IP addressing scheme between the nodes -- with one addressing scheme for each overlay network. One way this can be achieved is by assigning IP addresses to the tunnel endpoints (like illustrated above). Other, better alternatives are possible. However, this is outside the scope of this article so we will leave that as an exercise to the reader.

- In a public cloud environment this procedure can be scaled-out to large number of nodes by automating those steps during server boot-up process and then cloning multiple servers from the resulting server image.

3. Node roles and configuration

For our setup we will use a multi-node configuration, but with a bare minimum of OpenStack services needed for our purpose -- namely illustrating the OVS and Quantum network service functionality. The setup consists of 3 servers with the following roles:

- **Controller node**: As the name implies, it runs all OpenStack controler services: Keystone, Nova (nova-api, nova-scheduler), Glance (glance-api, glance-registry), and all the auxiliary systems (the MySQL databases, the RabbitMQ message queue, etc). Also, for storing Glance images we will be using Swift object storage running on this server.

  Please note that for our purpose we will not be using the GUI interface (project Horizon) nor run the Nova services that are necessary to do so (e.g. nova-consoleauth, nova-novncproxy, etc). Similarly, we will not be installing / configuring block storage services (Cinder) in our installation.

  Also, from a network / Quantum perspective, due to limitations in current L3 agent functionality we are unable to run "nova-metadata" service, or access instance metadata information (more on this below in the **Limitations** section). For more details on the service and the necessary configuration please see here. For information on how to otherwise enable Nova Metadata API with the current (Feb 2013) code base please see this blog post.

Please note that from an OVS / Quantum perspective the *only* component that runs on this server is "quantum-server" itself -- not the OVS plugin / OVS agent, nor any OVS components.

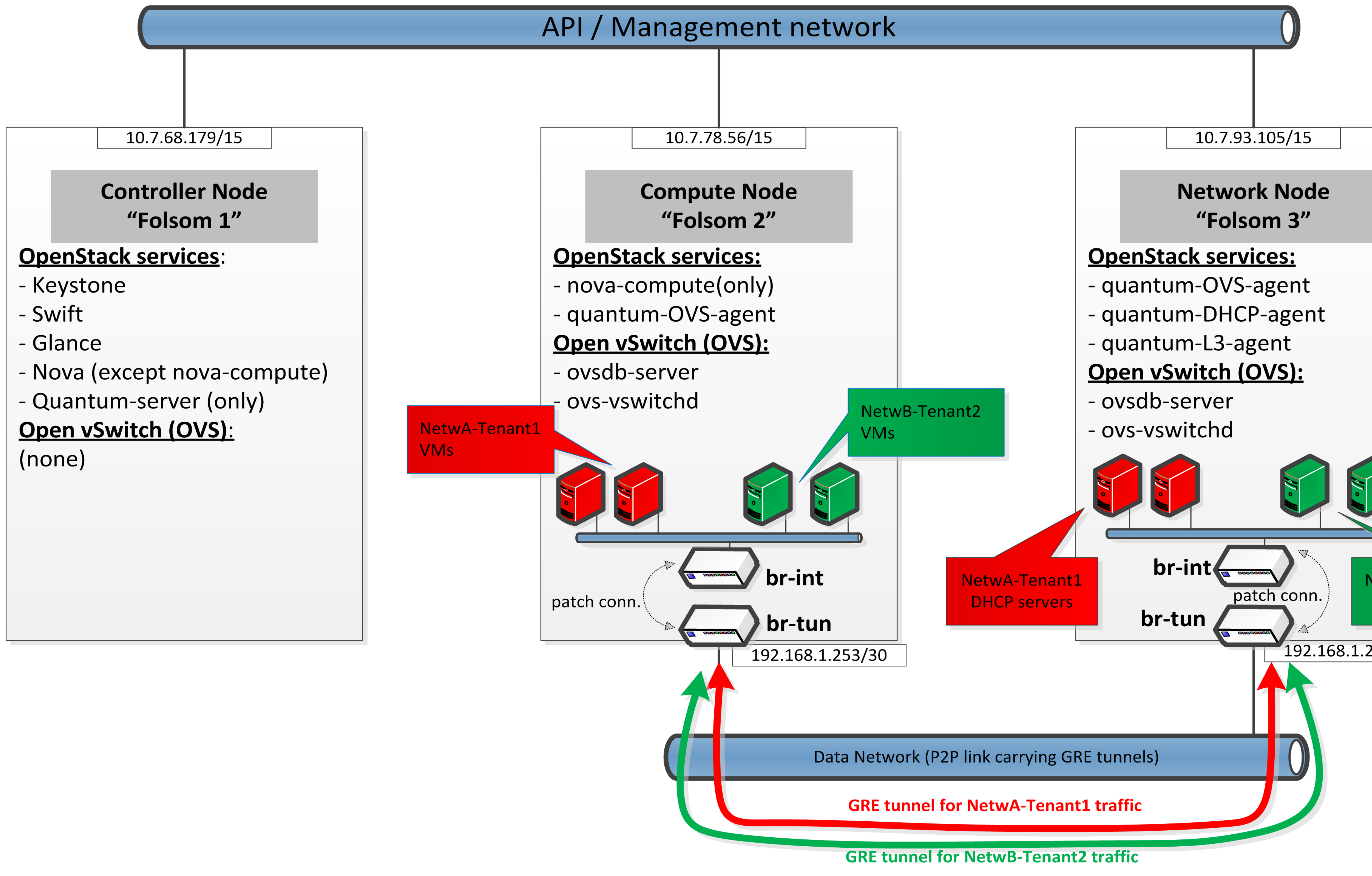
- **Compute node**: This is a server that has adequate capacity for hosting Nova VMs itself -- in our case, a "double extra large" instance on hpcloud.com. In addition to running "nova-compute" this server runs the Quantum plugin for OVS and the OVS agent (same component), as well as a local instance of OVS.
- **Network node**: The role of this node is to provide Nova VM instances with connectivity to external resources. In addition to running the Quantum OVS agent it runs the Quantum DHCP agent and Quantum L3 agent. In addition it also runs a local instance of OVS.

The servers are interconnected as follows:

- All three nodes are connected to the **API / management network**. As described above, in our case this network is instantiated using SSH-based VPNs by re-purposing the "private IP address" of the servers. However, this is completely transparent to the OpenStack services running on the servers.
- Additionally, the "Compute" and "Network" nodes are connected via a separate **Data network**. The role of this network is to provide Nova VM instances running on "Compute" node with external network connectivity via the "Network" node. In our particular setup this was instantiated as a point-to-point link between the different servers, but -- just as for the API / management network -- this is transparent for our purpose.

For the complete details on what types of networks are used in Quantum please see here.

Our setup is illustrated in the following picture:



For the remainder of this document we will detail the different components in our setup, their functionality and how they are configured. As we mentioned in the **Introduction** section, in the interest of brevity we will focus mostly on the network aspects (OVS and Quantum). However, for the sake of completeness we will include working configuration details for all the services necessary in the appendices.

4. Keystone, Swift, Glance and Nova -- Controller node

As the name implies the main function of the "Controller" node is to run the main OpenStack services (API endpoints). The following services are to be installed on the Controller node

4.1 Keystone

Keystone is used by Quantum for identifying the tenants, roles and users that create and manipulate the virtual networks in OpenStack. As a detailed installation guide for Keystone is not the main focus of this text please consult **Appendix A** for a synopsis of the installation steps and configuration files necessary to do so. Here we assume that Keystone is installed and running on the Controller node.

Once that achieved, we can now proceed in creating the tenants, roles and users that we will be using. Particularly we need create Keystone users for the OpenStack services that we will be deploying (Please note that all "keystone" commands below assume the "keystone" shell alias defined in **Appendix A** rather then the Keystone Python client CLI command itself):

```
keystone role-create --name=admin
keystone tenant-create --name=service
keystone user-create --name=swiftadmin --pass=swiftadminpasswd
keystone user-create --name=glanceadmin --pass=glanceadminpasswd
keystone user-create --name=novaadmin --pass=novaadminpasswd
keystone user-create --name=quantumadmin --pass=quantumadminpasswd
```

Then to each of those Keystone users we need to grant them the role of "admin" within the tenant "service". This can be achieved using the command:

```
keystone user-role-add --user_id <userID> --role_id <roleID-for-"admin"-role> --tenant_id <tenantID-for-"service"-tenant>
```

where "userID" will be, respectively, the user IDs returned by the "keystone user-create" commands above.

At this point we can now create the OpenStack services:

```
keystone service-create --name=keystone --type=identity --description="Keystone Identity Service"
keystone service-create --name=swift --type=object-store --description="Swift Object-store Service"
keystone service-create --name=glance --type=image --description="Glance Image Service"
keystone service-create --name=nova --type=compute --description="Nova Compute Service"
keystone service-create --name=quantum --type=network --description="Quantum Networking service"
```

Next is creating the API service endpoints. Note that below we will need to replace `<keystone-service_id>` `<swift-service_id>` `<glance-service_id>` `<nova-service_id>` and, respectively, `<quantum-service_id>` with the corresponding numerical Ids returned when we created the services (or, subsequently, as reported by the "`keystone service-list`" command):

```
keystone endpoint-create --region MyRegion --service_id <keystone-service_id> --publicurl 'http://10.7.68.179:5000/v2.0' --adminurl 'http://10.7
keystone endpoint-create --region MyRegion --service_id <swift-service_id> --publicurl 'http://10.7.68.179:8080/v1/AUTH_$(tenant_id)s' --adminur
keystone endpoint-create --region MyRegion --service_id <glance-service_id> --publicurl 'http://10.7.68.179:9292/v1' --adminurl 'http://10.7.68.
keystone endpoint-create --region MyRegion --service_id <nova-service_id> --publicurl 'http://10.7.68.179:8774/v2/$(tenant_id)s' --adminurl 'htt
keystone endpoint-create --region MyRegion --service_id <quantum-service-id> --publicurl 'http://10.7.68.179:9696' --adminurl 'http://10.7.68.17
```

To illustrate the multitenant capabilities in Quantum -- and follow the best practice -- we will create another tenant, called "endusers". Within that tenant we will be creating user "enduser1" (with "admin" privileges within tenant "endusers") that we will later use for instantiating VMs:

```
keystone tenant-create --name=endusers
keystone user-create --name=enduser1 --pass=enduserpasswd1
keystone user-role-add --user_id <userID-of-"enduser1"> --role_id <roleID-for-"admin"-role> --tenant_id <tenantID-for-"endusers"-tenant>
```

This concludes the Keystone configuration.

4.2 Swift with Keystone authentication

In our setup we use Swift solely for storing VM images i.e. as a Glance back-end. Since the servers we used in this article are VMs themselves (i.e. with only ephemeral storage as a local disks) for Swift object storage we used a persistent block storage device mounted on the Controller node.

As it is secondary to our purpose we leave detailed Swift installation and configuration outside the scope of the main text. Please see **Appendix B** for a synopsis of the necessary steps.

4.3 Glance with Swift back-end

In our setup we are using Glance for storing image files for Nova instances. The two Glance services that run on the controller are:

- Glance API server.
- Glance registry server, the component that records the image metadata in an SQL database. For the purpose of this blog post we will be using the same MySQLserver on the Controller node like the rest of the services in our installation.

Detailed Glance installation and configuration steps are not the main focus of this article. However, for completeness they are included in **Appendix C** below.

Once Glance is installed and configured we are ready to load some VM images. For the purpose of our setup we use the CirrOS Linux image - a minimalistic image to be used for testing purposes. Please note that all the commands below assume the "glance" shell alias configured in **Appendix C** (rather than the "glance" Python client command itself):

```
cd /mnt
wget https://launchpad.net/cirros/trunk/0.3.0/+download/cirros-0.3.0-x86_64-disk.img
glance add name="CirrOS-0.3.0-x86_64" is_public=true  container_format=bare disk_format=qcow2 distro="cirrOS-0.3.0-x86_64" < cirros-0.3.0-x86_64
```

The last step is a quick check -- via "glance index" and "glance show" commands -- that the image was uploaded correctly and is ready for provisioning. The ouptut of these commands should be similar to the following:

```
root@Folsom1:~# glance index
ID                                   Name                           Disk Format          Container Format     Size
------------------------------------ ------------------------------ -------------------- -------------------- --------------
7789ebb5-14e8-4c44-a4ea-1bffdc52df51 CirrOS-0.3.0-x86_64            qcow2                bare                        9761280

root@Folsom1:~# glance show 7789ebb5-14e8-4c44-a4ea-1bffdc52df51
URI: http://10.7.68.179:9292/v1/images/7789ebb5-14e8-4c44-a4ea-1bffdc52df51
Id: 7789ebb5-14e8-4c44-a4ea-1bffdc52df51
Public: Yes
Protected: No
Name: CirrOS-0.3.0-x86_64
Status: active
Size: 9761280
Disk format: qcow2
Container format: bare
Minimum Ram Required (MB): 0
Minimum Disk Required (GB): 0
Owner: 8032be08210a424687ff3622338ffe23
Property 'distro': cirrOS-0.3.0-x86_64
Created at: 2012-12-16T15:37:34
Updated at: 2012-12-16T15:37:34
```

4.4 Nova on the Controller node -- nova-api and nova-scheduler

The only two Nova services we will be installing on the Controller server are the "nova-api" and "nova-scheduler". The astute reader may note that a notable exception is "nova-api-metada", the Metadata service (please see here for more information about the Metadata service itself. Please also consult this article for instructions on how the Metadata service is to be configured when using the "Folsom" release of OpenStack services with the "Grizzly" code base for Quantum -- which is the very configuration we are using in this article as well).

The reason for this omission is the following: One critical requirement for the Metadata service is that our VM instances must be access the API endpoint (IP address + port) where the Metadata service is located (please see the notes here and here). However, due to a limitation in the current version of Quantum L3 agent and particularities of our setup that is not possible. Please see **Limitations** section below for more details.

While this does impose some limitations on our setup -- e.g. we are unable to use "Ubuntu" Linux images for our setup as they require access to the Metadata service for generating SSH keys -- we consider this is a secondary issue given the focus of this article.

Also, as noted in the **Node roles and configuration** section we will not be installing / configuring any Nova services necessary for the graphical dashboard (project Horizon) e.g. "nova-console" "nova-consoleauth", "nova-novncproxy", etc.

In our case, the Nova installation the Controller node proceeds as follows:

```
apt-get -y  install  nova-common nova-api nova-doc nova-scheduler python-novaclient

# Stop the services that are spawned at postinstall
/etc/init.d/nova-api stop
/etc/init.d/nova-scheduler stop

# First, we configure the MySQL database used for Nova and create a dedicated user to access it.
# The database is located on the same MySQL instance used by all services in our setup
mysql -u root -pmysqlpasswd -h 10.7.68.179 -e "create database nova;"
mysql -u root -pmysqlpasswd -h 10.7.68.179 -e "create user 'novadbadmin'@'%' identified by 'novadbadminpasswd';"
mysql -u root -pmysqlpasswd -h 10.7.68.179 -e "grant all privileges on nova.* to 'novadbadmin'@'%';"
```

The only Nova configuration on the Controller node is in "/etc/nova/nova.conf" "/etc/nova/api-paste.ini" and, respectively, "/etc/nova/policy.json":

```
######## Nova configuration file "/etc/nova/nova.conf"

cat <<NOVACNF >/etc/nova/nova.conf
[DEFAULT]
```

```
# LOGS/STATE
verbose=True
logdir=/var/log/nova
state_path=/var/lib/nova
lock_path=/var/lock/nova

# AUTHENTICATION
auth_strategy=keystone

# DATABASE
sql_connection=mysql://novadbadmin:novadbadminpasswd@10.7.68.179/nova

# COMPUTE
connection_type=libvirt
libvirt_type=qemu
instance_name_template=VM-%08x
api_paste_config=/etc/nova/api-paste.ini
allow_resize_to_same_host=True
libvirt_vif_driver=nova.virt.libvirt.vif.LibvirtOpenVswitchDriver
libvirt_use_virtio_for_bridges=True

# APIs -- Only enable compute and metadata (without EC2 compat)
enabled_apis = osapi_compute,metadata
osapi_compute_extension=nova.api.openstack.compute.contrib.standard_extensions

# RABBITMQ
rabbit_host=10.7.68.179
rabbbit_user=guest
rabbit_password=guest

# SCHEDULER
compute_scheduler_driver=nova.scheduler.filter_scheduler.FilterScheduler

# GLANCE
image_service=nova.image.glance.GlanceImageService
glance_api_servers=10.7.68.179:9292

# NETWORK -- Quantum
network_api_class=nova.network.quantumv2.api.API
quantum_url=http://10.7.68.179:9696
quantum_auth_strategy=keystone
quantum_admin_tenant_name=service
quantum_admin_username=quantumadmin
quantum_admin_password=quantumadminpasswd
quantum_admin_auth_url=http://10.7.68.179:5001/v2.0/

# NOVNC CONSOLE
# Only serial
vnc_enabled=False

NOVACNF

######## Nova configuration file "/etc/nova/api-paste.ini"

cat <<NOVAPASTE >/etc/nova/api-paste.ini
############
# Metadata #
############
[composite:metadata]
use = egg:Paste#urlmap
/: meta

[pipeline:meta]
pipeline = metaapp

[app:metaapp]
paste.app_factory = nova.api.metadata.handler:MetadataRequestHandler.factory

#############
# Openstack #
#############

[composite:osapi_compute]
use = call:nova.api.openstack.urlmap:urlmap_factory
/: oscomputeversions
/v1.1: openstack_compute_api_v2
```

```
/v2: openstack_compute_api_v2

[composite:openstack_compute_api_v2]
use = call:nova.api.auth:pipeline_factory
noauth = faultwrap noauth ratelimit osapi_compute_app_v2
deprecated = faultwrap auth ratelimit osapi_compute_app_v2
keystone = faultwrap authtoken keystonecontext ratelimit osapi_compute_app_v2
keystone_nolimit = faultwrap authtoken keystonecontext osapi_compute_app_v2

[composite:openstack_volume_api_v1]
use = call:nova.api.auth:pipeline_factory
noauth = faultwrap noauth ratelimit osapi_volume_app_v1
deprecated = faultwrap auth ratelimit osapi_volume_app_v1
keystone = faultwrap authtoken keystonecontext ratelimit osapi_volume_app_v1
keystone_nolimit = faultwrap authtoken keystonecontext osapi_volume_app_v1
[filter:faultwrap]
paste.filter_factory = nova.api.openstack:FaultWrapper.factory

[filter:auth]
paste.filter_factory = nova.api.openstack.auth:AuthMiddleware.factory

[filter:noauth]
paste.filter_factory = nova.api.openstack.auth:NoAuthMiddleware.factory

[filter:ratelimit]
paste.filter_factory = nova.api.openstack.compute.limits:RateLimitingMiddleware.factory

[app:osapi_compute_app_v2]
paste.app_factory = nova.api.openstack.compute:APIRouter.factory

[pipeline:oscomputeversions]
pipeline = faultwrap oscomputeversionapp

[app:osapi_volume_app_v1]
paste.app_factory = nova.api.openstack.volume:APIRouter.factory

[app:oscomputeversionapp]
paste.app_factory = nova.api.openstack.compute.versions:Versions.factory

[pipeline:osvolumeversions]
pipeline = faultwrap osvolumeversionapp

[app:osvolumeversionapp]
paste.app_factory = nova.api.openstack.volume.versions:Versions.factory

##########
# Shared #
##########

[filter:keystonecontext]
paste.filter_factory = nova.api.auth:NovaKeystoneContext.factory

[filter:authtoken]
paste.filter_factory = keystone.middleware.auth_token:filter_factory
service_protocol = http
service_host = 10.7.68.179
service_port = 5000
auth_protocol = http
auth_host = 10.7.68.179
auth_port = 5001
admin_tenant_name = service
admin_user = novaadmin
admin_password = novaadminpasswd
delay_auth_decision = 0
signing_dir = /etc/nova

NOVAPASTE

######## Nova configuration file "/etc/nova/policy.json"

cat <<NOVAPOLICY >/etc/nova/policy.json
{
    "admin_or_owner":  [["role:admin"], ["project_id:%(project_id)s"]],
    "default": [["rule:admin_or_owner"]],

    "compute:create": [],
    "compute:create:attach_network": [],
    "compute:create:attach_volume": [],
    "compute:get_all": [],

    "admin_api": [["role:admin"]],
    "compute_extension:accounts": [["rule:admin_api"]],
    "compute_extension:admin_actions": [["rule:admin_api"]],
    "compute_extension:admin_actions:pause": [["rule:admin_or_owner"]],
    "compute_extension:admin_actions:unpause": [["rule:admin_or_owner"]],
    "compute_extension:admin_actions:suspend": [["rule:admin_or_owner"]],
    "compute_extension:admin_actions:resume": [["rule:admin_or_owner"]],
    "compute_extension:admin_actions:lock": [["rule:admin_api"]],
    "compute_extension:admin_actions:unlock": [["rule:admin_api"]],
    "compute_extension:admin_actions:resetNetwork": [["rule:admin_api"]],
    "compute_extension:admin_actions:injectNetworkInfo": [["rule:admin_api"]],
    "compute_extension:admin_actions:createBackup": [["rule:admin_or_owner"]],
    "compute_extension:admin_actions:migrateLive": [["rule:admin_api"]],
    "compute_extension:admin_actions:migrate": [["rule:admin_api"]],
    "compute_extension:aggregates": [["rule:admin_api"]],
    "compute_extension:certificates": [],
    "compute_extension:cloudpipe": [["rule:admin_api"]],
    "compute_extension:console_output": [],
    "compute_extension:consoles": [],
    "compute_extension:createserverext": [],
    "compute_extension:deferred_delete": [],
    "compute_extension:disk_config": [],
    "compute_extension:extended_server_attributes": [["rule:admin_api"]],
    "compute_extension:extended_status": [],
    "compute_extension:flavorextradata": [],
    "compute_extension:flavorextraspecs": [],
    "compute_extension:flavormanage": [["rule:admin_api"]],
    "compute_extension:floating_ip_dns": [],
    "compute_extension:floating_ip_pools": [],
    "compute_extension:floating_ips": [],
    "compute_extension:hosts": [["rule:admin_api"]],
    "compute_extension:keypairs": [],
    "compute_extension:multinic": [],
    "compute_extension:networks": [["rule:admin_api"]],
    "compute_extension:quotas": [],
    "compute_extension:rescue": [],
    "compute_extension:security_groups": [],
    "compute_extension:server_action_list": [["rule:admin_api"]],
    "compute_extension:server_diagnostics": [["rule:admin_api"]],
    "compute_extension:simple_tenant_usage:show": [["rule:admin_or_owner"]],
    "compute_extension:simple_tenant_usage:list": [["rule:admin_api"]],
    "compute_extension:users": [["rule:admin_api"]],
    "compute_extension:virtual_interfaces": [],
    "compute_extension:virtual_storage_arrays": [],
    "compute_extension:volumes": [],
    "compute_extension:volumetypes": []

}
NOVAPOLICY
```

At this point we can start the Nova services on the Controller node:

```
# Before the first time we start the Nova services we need to create the Nova DB tables
nova-manage db sync

# First Nova service to start is "nova-api". As we start it in debug mode it will not detach from the terminal
nova-api --debug |& tee /var/log/nova/nova-api.log

#Second is nova-scheduler. Will also lock the terminal in debugging mode
nova-scheduler --debug |& tee /var/log/nova/nova-scheduler.log
```

This concludes the installation and configuration instructions for the Controller node.

5. Nova-compute -- Compute node

The only Nova service running on the Compute node is "nova-compute". Here is a brief description how to install nova-compute on this node (please note this is assuming that the "common prerequisites" configuration detailed in **Appendix A** are performed first):

```
# Our server is a VM, so we will be using "qemu-kvm" as virtualization layer
apt-get -y install qemu-kvm libvirt-bin

# Install "nova-compute"
apt-get -y install nova-common nova-compute nova-compute-kvm python-nova python-novaclient
```

In terms of Nova configuration, the files we are using -- "/etc/nova/nova.conf" , "/etc/nova/api-paste.ini" and, respectively, "/etc/nova/policy.json" -- are identical with the ones on the Controller node. As such please refer to previous section for how to configure those. Once that is done we can start "nova-compute" on this server:

```
# First make sure everything will be in the right place
mkdir -p /var/log/nova /var/lock/nova/ /var/lib/nova

# Start "nova-compute". As debugging is turned on it will lock the terminal
nova-compute --debug |& tee /var/log/nova/nova-compute.log
```

This concludes the Nova installation and configuration steps for the Compute node.

6. Open vSwitch -- Bootstrapping from source code

The configuration steps outlined in this section are common to both the Compute node and Network node. As our focus is to illustrate functionality rather than providing extensive installation instructions we are including those steps only once below. However, for a functioning installation they are to be duplicated for both the Controller node and Network node in our setup.

While the purpose of this document is explain how Open vSwitch (OVS) and Quantum operate to provide virtual networking services in OpenStack rather than Open vSwitch itself, a brief introduction to OVS is in order.

As the OVS homepage states: "Open vSwitch is a production quality, multilayer virtual switch licensed under the open source Apache 2.0 license". The functionality that OVS provides is split between two mechanisms:

- A "Fast Datapath" kernel module. As the name implies, its role is fast forwarding of packets with minimum overhead.
- A controller module running in user-space. Its role is to receive configuration information -- either directly (via configuration commands) or programatically (from external entities) -- and then instruct the kernel model on the appropriate action, as follows.

Under normal operations the kernel module forwards network traffic based on the configuration information it has locally. For any network traffic flows for which kernel module does not have local configuration information (initially: any traffic) it forwards the packets to the controller. The controller consults its database then instructs the kernel module on what actions to take for traffic conforming to flow specifications that match those packets. After this the kernel module will save that flow specification as local configuration information and will subsequently forward any network traffic matching that flow specification without consulting the controller.

It is outside the scope of this article to discuss in-depth what consists a "network flow", how network flows are defined and instantiated. Please see here and here for two excellent resources on different types of network flows ("coarse" vs "fine", "proactive" vs "reactive", etc.) and otherwise excellent information source on OpenFlow and SDN. This blog post is another resource detailing the OVS concepts we mentioned above and how OpenFlow can be used as a programatic interface to OVS.

In terms of software components our OVS installation will use the following elements:

- The "openvswitch.ko" -- and, for compatibility purposes, "brcompat.ko" -- kernel modules. These implement the "Fast Datapath" modules mentioned above.
- "ovs-vswitchd" is the controller component that runs as a server in user-space.
- "ovsdb-server" is a JSON database server which contains the configuration for the controller program. It serves this database via JSON RPC. The database can be accessed (i.e. OVS can be configured) either locally (via UNIX socket) or remotely (via TLS/SSL)
- Various tools and utilities provided as part of the OVS distribution: "ovs-dpctl" for controlling the kernel datapath module; "ovs-vsctl" for direct configuration (via UNIX socket); "ovs-ofctl" for querying and controlling network flow (OpenFlow) configuration

For our installation we will be using "trunk" -- the current Open vSwitch code base as provided by the official OVS software repository:

```
# Get some prerequisites
apt-get -y install git  python-simplejson python-qt4 python-zopeinterface  python-twisted-conch automake autoconf gcc uml-utilities libtool  pkg
onfig make libssl-dev iproute tcpdump linux-headers-`uname -r`

# If needed change this to something more appropriate
mkdir -p /usr/src/OVS-`date +%Y%m%d`
cd /usr/src/OVS-`date +%Y%m%d`

# Clone OVS trunk in there
git clone git://openvswitch.org/openvswitch
cd openvswitch

# Build and install OVS
./boot.sh
./configure --with-linux=/lib/modules/`uname -r`/build --prefix="/"
make |& tee make.out
make install |& tee install.out

# Load the OVS kernel modules
insmod  ./datapath/linux/openvswitch.ko
insmod  ./datapath/linux/brcompat.ko

# Make sure we have a clean system before putting configuration files in place
rm -fr /etc/openvswitch/ /var/run/openvswitch
mkdir -p /etc/openvswitch
mkdir -p /var/run/openvswitch

# Create an initial, empty JSON configuration database located in "/etc/openvswitch/conf.db" using the schema in "vswitch.ovsschema"
ovsdb-tool create /etc/openvswitch/conf.db vswitchd/vswitch.ovsschema

# Start the JSON database server. Please note that this will lock your terminal and output verbose debugging info
ovsdb-server /etc/openvswitch/conf.db --remote=punix:/var/run/openvswitch/db.sock --remote=db:Open_vSwitch,manager_options --private-key=db:SSL,

#Initialize the database.
ovs-vsctl --no-wait init

# Start the OVS controller, "vswitchd"
ovs-vswitchd -v
```

After these steps we have a running instance of OVS configured for local access. We can verify that using the "ovs-vsctl" command. In our case after installing OVS on the Compute node this results in:

```
root@Folsom2:~# ovs-vsctl -v --version
ovs-vsctl (Open vSwitch) 1.7.1
Compiled Dec 18 2012 13:05:42

root@Folsom2:~# ovs-vsctl -v show
2013-01-14T08:37:53Z|00001|reconnect|DBG|unix:/usr/local/var/run/openvswitch/db.sock: entering BACKOFF
2013-01-14T08:37:53Z|00002|reconnect|INFO|unix:/usr/local/var/run/openvswitch/db.sock: connecting...
2013-01-14T08:37:53Z|00003|reconnect|DBG|unix:/usr/local/var/run/openvswitch/db.sock: entering CONNECTING
2013-01-14T08:37:53Z|00004|poll_loop|DBG|wakeup due to [POLLOUT] on fd 4 (<->/usr/local/var/run/openvswitch/db.sock) at lib/stream-fd.c:139
2013-01-14T08:37:53Z|00005|reconnect|INFO|unix:/usr/local/var/run/openvswitch/db.sock: connected
2013-01-14T08:37:53Z|00006|reconnect|DBG|unix:/usr/local/var/run/openvswitch/db.sock: entering ACTIVE
2013-01-14T08:37:53Z|00007|jsonrpc|DBG|unix:/usr/local/var/run/openvswitch/db.sock: send request, method="monitor", params=["Open_vSwitch",null,
2013-01-14T08:37:53Z|00008|poll_loop|DBG|wakeup due to [POLLIN] on fd 4 (<->/usr/local/var/run/openvswitch/db.sock) at lib/stream-fd.c:143
2013-01-14T08:37:53Z|00009|jsonrpc|DBG|unix:/usr/local/var/run/openvswitch/db.sock: received reply, result={"Open_vSwitch":{"8aa4bbf9-e536-4c23-
8aa4bbf9-e536-4c23-aaa5-5d81ba16073b
```

At this point we can configure OVS -- either directly using "ovs-vsctl" commands or programatically using e.g. OpenFlow. Please see this article or this screencast for details on how to perform the latter using Floodlight SDN controller, which can also run as a network backend for OpenStack via its Quantum plugin. However, for the remainder of this OVS will be configured automatically, as part of the OVS Quantum plugin configuration below.

7. Quantum -- Overview, Functionality, Installation and configuration

To quote its official Wiki page "Quantum is an OpenStack project to provide 'networking as a service' between interface devices (e.g., vNICs) managed by other Openstack services (e.g. Nova)".

In order to accommodate different types of network devices and network functions Quantum is designed as a "pluggable" framework that abstracts the functionality provided by different back-ends -- called "plugins"-- via a single, extensible REST API layer. This design allows providing networking services using various technologies -- virtual or physical switches -- from different vendors via a clean, unified network service abstraction layer.

Reflecting the "pluggable framework" design the following components can be distinguished from a service architecture point of view:

- The Quantum Server (aka "controller"). It's the component that implements the Quantum REST API. The sever is responsible for passing the API calls to/from the back-end plugin for processing.
- The "plugin". It is the device specific back-end that implements the functionality exposed by the API layer. At the time of this writing there are multiple plugins included in the Quantum software distribution, both vendor specific and vendor-neutral -- please see here for the current list. However, in this document we will be focusing on the Open vSwitch Quantum plugin.

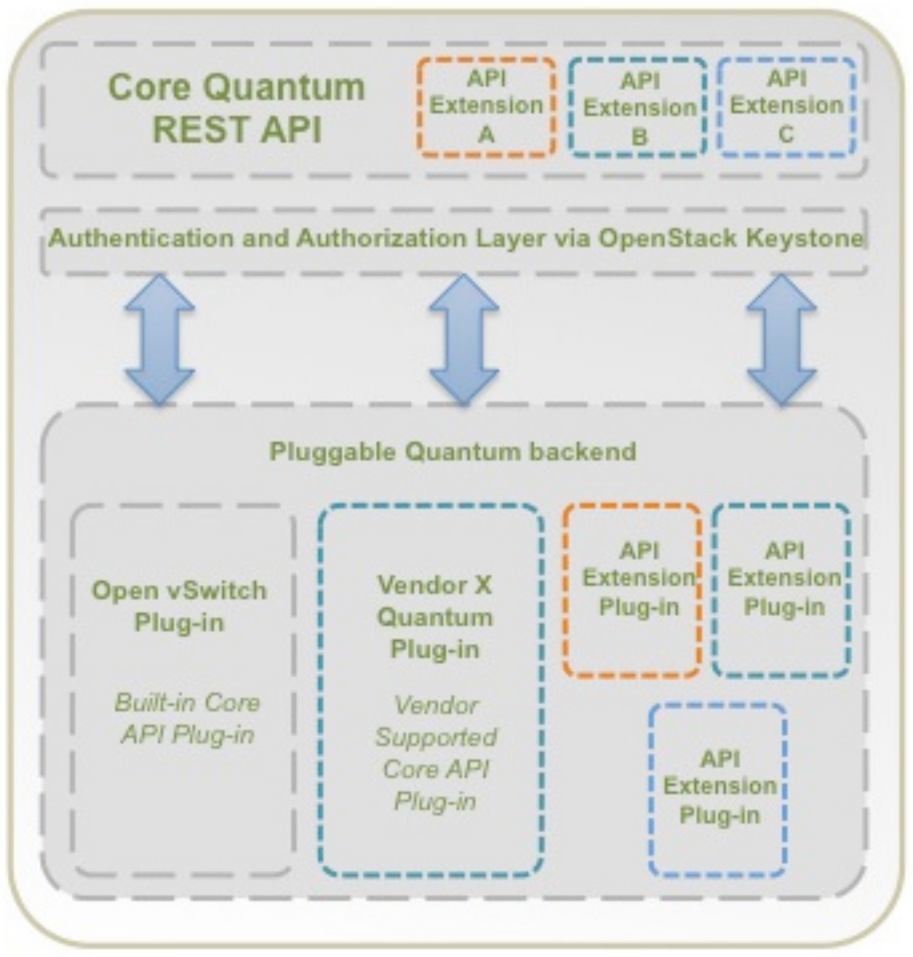
  The plugin is responsible for processing the API calls it receives from the Quantum server and for storing the resulting Quantum networking constructs, which are maintained in a separate database. One important thing to note here is that this database is used to store Quantum network models and constructs and is thus logically distinct to any configuration information local to the plugin itself. For example in the OVS plugin this is a MySQL database, which is different from the JSON database that OVS is using locally for storing its own configuration information.
- The plugin agent(s). These are the components that run on all "Compute" nodes (i.e. nodes running local VM instances) and all "Network" nodes (providing connectivity to external resources to those VM instances). The plugin agents are in charge of handling all the network traffic to / from VM instances according the the networking models implemented by the plugin and segragating between the different types of traffic.

  One thing to note here is that in the case of the OVS plugin the "plugin" and "plugin agent" are currently configured via the same configuration file -- namely "/etc/quantum/plugins/openvswitch/ovs_quantum_plugin.ini". However the two roles logically distinct, and hence the content of the file will be different on different nodes depending on their role. Please see below for more details.

- Quantum agents. These are plugin-independent components for providing additional networking services. The two Quantum agents that are currently provided with the Quantum software distribution is the Quantum DHCP agent (providing DHCP services to VM instances) and Quantum L3 agent (used for providing VM instances with L3 connectivity to external resources).

  Please note that despite the name similarity these "Quantum agents" are distinct from the "plugin agents" above.

The Quantum service architecture and interaction with other OpenStack services (e.g. Keystone, for Authentication and Authorization) can be illustrated as follows (image courtesy of Somik) :



Quantum Cloud Networking Fabric Architecture

7.1 Bootstrapping Quantum from source code. Common configuration

Next step is to install Quantum. For the purpose of this guide we will be installing the most current version of the source code as available from the Quantum official code repository on GitHub. This step is identical for all the nodes in our setup -- and as such needs to be replicated on all the nodes -- but we are including it here only once:

```
######## First make sure we have the right preqs. in place
apt-get -y install git python-pip  python-setuptools python-dev libxslt1-dev automake autoconf gcc make

# If needed change this to something more suitable to your setup
mkdir -p /usr/src/quantum-`date +%Y%m%d`
cd /usr/src/quantum-`date +%Y%m%d`

######### Get and install the Quantum python client
git clone https://github.com/openstack/python-quantumclient.git
cd python-quantumclient
pip install -r tools/pip-requires
python ./setup.py build 2>&1 | tee build.out
python ./setup.py install 2>&1 | tee install.out

cd ..

######### Get and install Quantum itself
git clone https://github.com/openstack/quantum.git
cd quantum/
pip install -r tools/pip-requires
python ./setup.py build 2>&1 | tee build.out
python ./setup.py install 2>&1 | tee install.out

cd ..
```

Also common to all nodes in our setup is Quantum (API) server configuration, contained in "/etc/quantun/quantum.conf", "/etc/quantum/policy.json" and "/etc/quantum/api-paste.ini":

```
# Firs make sure these exist so things can go in the right place
mkdir -p /etc/quantum/plugins/openvswitch /var/log/quantum

######## /etc/quantum/quantum.conf
cat <<QUANTUMCNF >/etc/quantum/quantum.conf
[DEFAULT]
verbose = True
```

```
# Show debugging output in logs (sets DEBUG log level output)
debug = True

# Address of the API server
bind_host = 10.7.68.179

# The port the API server binds to
bind_port = 9696

# Quantum plugin provider module
core_plugin=quantum.plugins.openvswitch.ovs_quantum_plugin.OVSQuantumPluginV2

# Paste configuration file
api_paste_config = api-paste.ini

# The strategy to be used for auth.
# Supported values are 'keystone'(default), 'noauth'.
auth_strategy = keystone

# AMQP exchange to connect to if using RabbitMQ or QPID
control_exchange = quantum

# If passed, use a fake RabbitMQ provider
fake_rabbit = False

# RabbitMQ settings
rabbit_host = 10.7.68.179
rabbit_port = 5672
rabbit_use_ssl = false
rabbit_userid = guest
rabbit_password = guest
rabbit_virtual_host = /
rabbit_max_retries = 0
rabbit_retry_interval = 1

# ============ Notification System Options =====================
# Notifications can be sent when network/subnet/port are create, updated or deleted.
# There are three methods of sending notifications: logging (via the
# log_file directive), rpc (via a message queue) and
# noop (no notifications sent, the default)
# RPC driver. DHCP agents needs it.
notification_driver = quantum.openstack.common.notifier.rpc_notifier

# default_notification_level is used to form actual topic name(s) or to set logging level
default_notification_level = INFO

# Defined in rpc_notifier, can be comma separated values.
# The actual topic names will be %s.%(default_notification_level)s
notification_topics = notifications

QUANTUMCNF

######## /etc/quantum/policy.json
cat <<QUANTUMPOLICYJSON >/etc/quantum/policy.json
{
    "admin_or_owner": "role:admin or tenant_id:%(tenant_id)s",
    "admin_or_network_owner": "role:admin or tenant_id:%(network_tenant_id)s",
    "admin_only": "role:admin",
    "regular_user": "",
    "shared": "field:networks:shared=True",
    "external": "field:networks:router:external=True",
    "default": "rule:admin_or_owner",

    "extension:provider_network:view": "rule:admin_only",
    "extension:provider_network:set": "rule:admin_only",

    "extension:router:view": "rule:regular_user",
    "extension:router:set": "rule:admin_only",
    "extension:router:add_router_interface": "rule:admin_or_owner",
    "extension:router:remove_router_interface": "rule:admin_or_owner",

    "extension:port_binding:view": "rule:admin_only",
    "extension:port_binding:set": "rule:admin_only",

    "subnets:private:read": "rule:admin_or_owner",
    "subnets:private:write": "rule:admin_or_owner",
    "subnets:shared:read": "rule:regular_user",
    "subnets:shared:write": "rule:admin_only",

    "create_subnet": "rule:admin_or_network_owner",
    "get_subnet": "rule:admin_or_owner or rule:shared",
    "update_subnet": "rule:admin_or_network_owner",
    "delete_subnet": "rule:admin_or_network_owner",

    "create_network": "",
    "get_network": "rule:admin_or_owner or rule:shared or rule:external",
    "create_network:shared": "rule:admin_only",
    "create_network:router:external": "rule:admin_only",
    "update_network": "rule:admin_or_owner",
    "delete_network": "rule:admin_or_owner",

    "create_port": "",
    "create_port:mac_address": "rule:admin_or_network_owner",
    "create_port:fixed_ips": "rule:admin_or_network_owner",
    "get_port": "rule:admin_or_owner",
    "update_port": "rule:admin_or_owner",
    "update_port:fixed_ips": "rule:admin_or_network_owner",
    "delete_port": "rule:admin_or_owner",

    "extension:service_type:view_extended": "rule:admin_only",
    "create_service_type": "rule:admin_only",
    "update_service_type": "rule:admin_only",
    "delete_service_type": "rule:admin_only",
    "get_service_type": "rule:regular_user"
}
QUANTUMPOLICYJSON

######## /etc/quantum/api-paste.ini
cat <<QUANTUMAPIPASTE >/etc/quantum/api-paste.ini
[composite:quantum]
use = egg:Paste#urlmap
/: quantumversions
/v2.0: quantumapi_v2_0

[composite:quantumapi_v2_0]
use = call:quantum.auth:pipeline_factory
noauth = extensions quantumapiapp_v2_0
keystone = authtoken keystonecontext extensions quantumapiapp_v2_0

[filter:keystonecontext]
paste.filter_factory = quantum.auth:QuantumKeystoneContext.factory

[filter:authtoken]
paste.filter_factory = keystoneclient.middleware.auth_token:filter_factory
service_protocol = http
service_host = 10.7.68.179
service_port = 5000
auth_protocol = http
auth_host = 127.0.0.1
auth_port = 5001
admin_tenant_name = service
admin_user = quantumadmin
admin_password = quantumadminpasswd
delay_auth_decision = 0
signing_dir = /etc/quantum

[filter:extensions]
paste.filter_factory = quantum.api.extensions:plugin_aware_extension_middleware_factory
[app:quantumversions]
paste.app_factory = quantum.api.versions:Versions.factory

[app:quantumapiapp_v2_0]
paste.app_factory = quantum.api.v2.router:APIRouter.factory

QUANTUMAPIPASTE
```

A couple of things to note in this configuration:

- As those configuration files are shared by all the nodes in our setup we need to specify the address of the Quantum server explicitly on all nodes (via the "bind_host" parameter). Particularly, it is the Quantum API server address on the API / management network.
- The back-end plugin is specified via the "core_plugin" parameter in the server configuration file "/etc/quantum/quantum.conf".
- Quantum is using Keystone as its authentication and authorization method. In order to access the Keystone service with "admin" privileges we will be using the

"quantumadmin" username we created in Keystone when we defined services, users, tenants and roles (above).

7.2 Quantum server -- Controller node

The only Quantum component running on the Controller node is the Quantum (API) server. Before configuring the Quantum server we need to create MySQL database that is used by the plugin to store the Quantum constructs. In our setup this database will be located on the Controller node:

```
mysql -u root -pmysqlpasswd -h 10.7.68.179 -e "create database ovs_quantum;"
mysql -u root -pmysqlpasswd -h 10.7.68.179 -e "create user 'quantumdbadmin'@'127.0.0.1' identified by 'quantumdbadminpasswd';"
mysql -u root -pmysqlpasswd -h 10.7.68.179 -e "grant all privileges on ovs_quantum.* to 'quantumdbadmin'@'%';"
```

As already described the Quantum (API) server configuration is contained in the configuration files shared by all nodes. The only missing piece for the Controller node is the configuration of the back-end plugin -- located in "/etc/quantum/plugins/openvswitch/ovs_quantum_plugin.ini":

```
######## OVS plugin configuration file for Controller node
cat <<OVSCNF >/etc/quantum/plugins/openvswitch/ovs_quantum_plugin.ini
[DATABASE]
sql_connection = mysql://quantumdbadmin:quantumdbadminpasswd@10.7.68.179/ovs_quantum
sql_max_retries = -1
reconnect_interval = 2

[OVS]
enable_tunneling=True
tenant_network_type=gre
tunnel_id_ranges=1:1000
# When using tunneling this should be reset from the default "default:2000:3999" to empty list
network_vlan_ranges =
# only if node is running the agent -- i.e. NOT on the controller
# local_ip=<data-net-IP-address-of-node>
#
# These are the defaults. Listing them nonetheless
integration_bridge = br-int
tunnel_bridge = br-tun
#
# Peer patch port in integration bridge for tunnel bridge
int_peer_patch_port = patch-tun
# Peer patch port in tunnel bridge for integration bridge
tun_peer_patch_port = patch-int

[AGENT]
rpc = True
# Agent's polling interval in seconds
polling_interval = 2
# Use "sudo quantum-rootwrap /etc/quantum/rootwrap.conf" to use the real
# root filter facility.
# Change to "sudo" to skip the filtering and just run the comand directly
root_helper = sudo

OVSCNF
```

One thing worth observing is that some OpenStack / Quantum installation guides mandate running the OVS plugin agent -- "quantum-openvswitch-agent" -- on the API server as well. The confusion seems to stem from the fact that in case of OVS both the back-end plugin and the OVS agent use the same configuration file -- namely "/etc/quantum/plugins/openvswitch/ovs_quantum_plugin.ini" -- for both of these two roles. However, as we mentioned already, the two roles are logically distinct -- and hence the contents of the file will be different -- depending on the role of the node. As a case in point the "local_ip" variable is commented out in the configuration file for the Controller node (above) but will be used accordingly for the Compute and, respectively, the Network nodes. More details as we will discuss the OVS agent configuration for the Compute and Network nodes.

At this point we can start the Quantum API server on the Controller node by pointing to the server configuration file and the back-end plugin configuration files:

```
quantum-server --config-file /etc/quantum/quantum.conf --config-file /etc/quantum/plugins/openvswitch/ovs_quantum_plugin.ini  --debug |& tee /va
```

7.3 OVS agent and Nova configuration -- Compute node

The only Quantum component running on the Compute node is the OVS agent. But first, as this is the Compute Node, we need to instruct "nova-compute" to use Quantum for VM instance networking services. Particularly in our setup in the "/etc/nova/nova.conf" file we specified in thet "/etc/nova/nova.conf" file that Nova is to be using Quantum APIs (via "network_api_class" variable) and, respectively, via "quantum_*" variables we instructed Nova where the Quantum service endpoint is located and what credentials to use for accessing that service. For the exact configuration options please see the content of "/etc/nova/nova.conf" in previous sections.

In addition to the Quantum server configuration files common to all nodes the OVS *agent* is configured via "/etc/quantum/plugins/openvswitch/ovs_quantum_plugin.ini":

```
cat <<OVSAGENTCOMP >/etc/quantum/plugins/openvswitch/ovs_quantum_plugin.ini
[DATABASE]
sql_connection = mysql://quantumdbadmin:quantumdbadminpasswd@10.7.68.179/ovs_quantum
sql_max_retries = -1
reconnect_interval = 2

[OVS]
enable_tunneling=True
tenant_network_type=gre
tunnel_id_ranges=1:1000
# When using tunneling this should be reset from the default "default:2000:3999" to empty list
network_vlan_ranges =
# only if node is running the agent -- i.e. NOT on the controller
local_ip=192.168.1.253

#
# These are the defaults. Listing them nonetheless
integration_bridge = br-int
tunnel_bridge = br-tun
#
# Peer patch port in integration bridge for tunnel bridge
int_peer_patch_port = patch-tun
# Peer patch port in tunnel bridge for integration bridge
tun_peer_patch_port = patch-int

[AGENT]
rpc = True
# Agent's polling interval in seconds
polling_interval = 2
# Use "sudo quantum-rootwrap /etc/quantum/rootwrap.conf" to use the real
# root filter facility.
# Change to "sudo" to skip the filtering and just run the comand directly
root_helper = sudo

OVSAGENTCOMP
```

A few comments:

- As we mentioned as we discussed Quantum service architecture, the role of the plugin agent is to ensure the network connectivity to/from VM instances according to the network models and constructs defined in Quantum. For the Compute node specifically, the OVS agent will be notified (by Nova) when a new VM instance is created so it can configure the underlying OVS accordingly.

- For "local_ip" we are specifying the Compute node endpoint on the Data network.

- In order to segregate the traffic between tenants and, within a tenant, the traffic between multiple networks defined for a tenant -- in other words for a given Tentant/Network combination -- Quantum can employ one of two mechanisms: VLANs (IEEE 802.1q tagging) or GRE tunnels.

  For the purpose of this guide we will be using GRE encapsulation. In this case for each Tenant/Network combination a GRE tunnel with an unique "tunnel-id" will be used to identify the network traffic belonging to that combination. As the name implies, the "tunnel_id_ranges" variable specifies the range of allowed GRE tunnel IDs.

For more information on how GRE tunnel IDs are employed, what the rest of the configuration options mean and how the OVS agent configures the underlying OVS to provide VM instance connectivity please see next section.

At this point we can start the OVS agent on the Compute node:

```
quantum-openvswitch-agent --config-file=/etc/quantum/quantum.conf --config-file=/etc/quantum/plugins/openvswitch/ovs_quantum_plugin.ini  |& tee
```

7.4 OVS agent and Quantum agents (DHCP, L3) -- Network node

As mentioned in Section 3, the role of the Network node is to provide VM instances access to additional services (e.g. DHCP) and to external resources. As such the following Quantum components will be running on the Network node:

- OVS agent. This will be the mirror of the agent running on the Compute node, connecting the Tenant/Network segregated VM network traffic to both Quantum services (DHCP, L3) and, respectively, to external network resources.
- Quantum DHCP agent. This component provides DHCP services to the VM instances on the Compute node in accordance to the network constructs instantiated in Quantum.
- Quantum L3 agent. As mentioned, this component defines how traffic to/from VM instances is to be routed to external resources.

Even in terms of configuration the OVS agent on the Network node mirrors the one on the Controller node: The two are identical except for the "local_ip" variable -- which here needs to reflect the IP address of the Network node on the Data network:

```
cat <<OVSAGENTNET >/etc/quantum/plugins/openvswitch/ovs_quantum_plugin.ini
[DATABASE]
sql_connection = mysql://quantumdbadmin:quantumdbadminpasswd@10.7.68.179/ovs_quantum
sql_max_retries = -1
reconnect_interval = 2

[OVS]
enable_tunneling=True
tenant_network_type=gre
tunnel_id_ranges=1:1000
# When using tunneling this should be reset from the default "default:2000:3999" to empty list
network_vlan_ranges =
# only if node is running the agent -- i.e. NOT on the controller
local_ip=192.168.1.254

#
# These are the defaults. Listing them nonetheless
integration_bridge = br-int
tunnel_bridge = br-tun
#
# Peer patch port in integration bridge for tunnel bridge
int_peer_patch_port = patch-tun
# Peer patch port in tunnel bridge for integration bridge
tun_peer_patch_port = patch-int

[AGENT]
rpc = True
# Agent's polling interval in seconds
polling_interval = 2
# Use "sudo quantum-rootwrap /etc/quantum/rootwrap.conf" to use the real
# root filter facility.
# Change to "sudo" to skip the filtering and just run the comand directly
root_helper = sudo

OVSAGENTNET
```

The configuration file for the Quantum DHCP agent is "/etc/quantum/dhcp_agent.ini":

```
cat <<QDHCPAGENT >/etc/quantum/dhcp_agent.ini
[DEFAULT]
# Show debugging output in log (sets DEBUG log level output)
debug = true

# Where to store dnsmasq state files.  This directory must be writable by the
# user executing the agent.
state_path = /var/run/dnsmasq


# The DHCP requires that an inteface driver be set.  Choose the one that best
# matches you plugin.
# OVS based plugins(OVS, Ryu, NEC, NVP, BigSwitch/Floodlight)
interface_driver = quantum.agent.linux.interface.OVSInterfaceDriver

# The agent can use other DHCP drivers.  Dnsmasq is the simplest and requires
# no additional setup of the DHCP server.
dhcp_driver = quantum.agent.linux.dhcp.Dnsmasq

# Allow overlapping IP (Must have kernel build with CONFIG_NET_NS=y and
# iproute2 package that supports namespaces).
# (!!!) OBS: This is "True" by default
use_namespaces = True

# Use "sudo quantum-rootwrap /etc/quantum/rootwrap.conf" to use the real
# root filter facility.
# Change to "sudo" to skip the filtering and just run the comand directly
root_helper = sudo

QDHCPAGENT
```

The only thing worth noting is that the Quantum DHCP agent is using Linux network namespaces by default, just like the Quantum L3 agent below. We will later discuss the impact and rationale.

The configuration for the Quantum L3 agent is in "/etc/quantum/l3_agent.ini":

```
cat <<L3AGENT >/etc/quantum/l3_agent.ini
[DEFAULT]
debug = True

# L3 requires that an interface driver be set.  Choose the one that best
# matches your plugin.
# OVS based plugins (OVS, Ryu, NEC) that supports L3 agent
interface_driver = quantum.agent.linux.interface.OVSInterfaceDriver

# The Quantum user information for accessing the Quantum API.
auth_url = http://10.7.68.179:5001/v2.0/
auth_strategy=keystone
auth_region = MyRegion
admin_tenant_name = service
admin_user = quantumadmin
admin_password = quantumadminpasswd

# Use "sudo quantum-rootwrap /etc/quantum/rootwrap.conf" to use the real
# root filter facility.
# Change to "sudo" to skip the filtering and just run the comand directly
root_helper = sudo

# Allow overlapping IP (Must have kernel build with CONFIG_NET_NS=y and
# iproute2 package that supports namespaces).
# If use_namespaces is set as False then the agent can only configure one router.
use_namespaces = True

# Where to store Quantum state files.  This directory must be writable by the
# user executing the agent.
state_path = /var/run/dnsmasq

# This is done by setting the specific router_id.
# router_id = d3b59445-3b19-4b0d-ba16-1099d81fba59

# Each L3 agent can be associated with at most one external network.  This
# value should be set to the UUID of that external network.  If empty,
# the agent will enforce that only a single external networks exists and
# use that external network id
# gateway_external_network_id =

# Indicates that this L3 agent should also handle routers that do not have
# an external network gateway configured.  This option should be True only
# for a single agent in a Quantum deployment, and may be False for all agents
# if all routers must have an external network gateway
# handle_internal_only_routers = True
# Name of bridge used for external network traffic. This should be set to
# empty value for the linux bridge
external_network_bridge = br-ex

# TCP Port used by Quantum metadata server
# metadata_port = 9697

# Send this many gratuitous ARPs for HA setup. Set it below or equal to 0
# to disable this feature.
# send_arp_for_ha = 3

# seconds between re-sync routers' data if needed
# periodic_interval = 40

# seconds to start to sync routers' data after
# starting agent
# periodic_fuzzy_delay = 5

L3AGENT
```

At this point we can start the OVS agent and, respectively, the Quantum DHCP and L3 agents. Please note that each of those commands will start in debugging mode and will lock the terminal, so they need to be started in their own shell:

```
quantum-openvswitch-agent --config-file=/etc/quantum/quantum.conf --config-file=/etc/quantum/plugins/openvswitch/ovs_quantum_plugin.ini  |& tee

quantum-dhcp-agent --config-file=/etc/quantum/quantum.conf --config-file=/etc/quantum/dhcp_agent.ini |& tee /var/log/quantum/quantum-dhcp-agent.

quantum-l3-agent --config-file=/etc/quantum/quantum.conf --config-file=/etc/quantum/l3_agent.ini |& tee /var/log/quantum/quantum-l3-agent.log
```

That concludes the Quantum configuration for the nodes in our setup. In the next section we will illustrate with working examples how those components interreact and explain the different Quantum concepts and constructs.

8. Illustrating Quantum concepts and constructs with working examples

As the official Quantum API documetation details, Quantum uses the following concepts and nomenclature:

- A **network** corresponds to a Layer 2 segment. It is the basic network topology construct in Quantum.
- A **subnet** is a block of IP addresses (IPv4 or IPv6). Basically, a CIDR prefix.
- A **port** is an attachment point for a device (physical or virtual).

In a nutshell what Quantum allows is programatic definition of "networks" on a per (Keystone) tenant basis and associating "networks" with one or several "subnets". In this manner when we are creating a VM instance we can specify to which such "network" the instance will be attached. While it is possible to create a VM with multiple NICs and specify how these will be attached to the different networks available to the tenant -- please see here how that is achieved -- the scope of the document is to illustrate the basic concepts themselves. As such -- and in order to keep matters simple -- we will only use a single NIC per VM in our examples.

As multiple networks can be defined for a tenant and each such network can be associated with one or several subnets we can have multiple tenants employing overlapping IP address spaces. In order to deal with this Quantum uses Provider Networks as a method for decoupling those constructs from the underlying network infrastructure (on one hand) and, on the other hand, segregate between them. While the Provider Extension to the Quantum API describes in detail the relevant concepts and the nomenclature, for the purpose of this guide it will suffice to say that we will be using GRE tunnels -- and, more precisely, GRE "tunnel-id"s -- as a method to differentiate the network traffic belonging to different Tenant/Network combinations ("Network A - Tenant 1" and, respectively, "Network 2 - Tenant B" in our setup diagram above).

One thing worth noting is that the way the term "provider" is used in Quantum documentation may be confusing. In the context above -- and in the CLI examples below -- the term "provider" is meant simply the mechanism for segregating between different types of network traffic -- in our case, GRE tunnels. This should not be confused with the term "provider network" used in Quantum nomenclature to denote access to external, non- OpenStack / Quantum controlled network resoruces i.e. external "providers" of network services. In order to be consistent with the CLI examples below we will use the term "provider" in its former meaning.

In order to illustrate how Tenant/Network combinations are isolated from each other in our setup we will be employing the Keystone tenant "endusers" and, respectively, tenant "services". The astute reader would be correct in objecting that the latter should be solely used for administrative purposes -- e.g. users "novaadmin", "glanceadmin" and, not the least, "quantumadmin" -- and not for "user" purposes (creating VMs, assigning VMs to networks, etc). However, in order to simplify things we will do so, but we will make a clear note on the context i.e. when we will use the "service" tenant credentials for administrative purposes versus when it is (improperly) used for "user" actions.

The first step in configuring Quantum is to create networks and subnets. As these constructs are assigned to different "tenants" these are actions that require "admin" credentials in Quantum (the commands below can be executed from any node in our setup since a part of the Quantum configuration common for all the nodes was installing the Python Quantum client. Also, **please note** that when we are including the shell prompt in the examples should serve as a clear indication that we are illustrating working examples, with input values and outputs valid for our setup. This as opposed to the configuration instructions in previous sections that need to be copied *ad literam*. YMMV.):

```
######## Enter the Quantum CLI using "admin" credentials
root@Folsom3:~# quantum --os_username quantumadmin --os_password quantumadminpasswd --os_tenant_name service --os_auth_url http://10.7.68.179:50
(quantum)

######## Create a network called "InternalNet1" for tenant "endusers". In our case, the Keystone tenant_id is 9be459eedc674635bbef86a47ac8896d
# Note that we will be using GRE as a "provider" mechanism and that traffic belonging to this network will be tagged with a GRE tunnel-id of "1"

(quantum) net-create --tenant_id 9be459eedc674635bbef86a47ac8896d InternalNet1 --provider:network_type gre  --provider:segmentation_id 1
Created a new network:
+---------------------------+--------------------------------------+
| Field                     | Value                                |
+---------------------------+--------------------------------------+
| admin_state_up            | True                                 |
| id                        | ff8a9831-3358-43e3-ae7f-bef03ff7bf6a |
| name                      | InternalNet1                         |
| provider:network_type     | gre                                  |
| provider:physical_network |                                      |
| provider:segmentation_id  | 1                                    |
| router:external           | False                                |
| shared                    | False                                |
| status                    | ACTIVE                               |
| subnets                   |                                      |
| tenant_id                 | 9be459eedc674635bbef86a47ac8896d     |
+---------------------------+--------------------------------------+

######## For the same tenant create a "subnet" with CIDR prefix of 172.16.1.0/24 , gateway 172.16.1.1 and associate this "subnet" with the "netw

(quantum) subnet-create --tenant_id 9be459eedc674635bbef86a47ac8896d --ip_version 4  InternalNet1 172.16.1.0/24 --gateway 172.16.1.1
Created a new subnet:
+------------------+--------------------------------------------+
| Field            | Value                                      |
+------------------+--------------------------------------------+
| allocation_pools | {"start": "172.16.1.2", "end": "172.16.1.254"} |
| cidr             | 172.16.1.0/24                              |
| dns_nameservers  |                                            |
| enable_dhcp      | True                                       |
| gateway_ip       | 172.16.1.1                                 |
| host_routes      |                                              |
| id               | 099c2fc9-3992-4963-bfca-4bb1af8fdeea         |
| ip_version       | 4                                            |
| name             |                                              |
| network_id       | ff8a9831-3358-43e3-ae7f-bef03ff7bf6a         |
| tenant_id        | 9be459eedc674635bbef86a47ac8896d             |
+------------------+----------------------------------------------+
```

One thing to note for the `172.16.1.0/24` subnet is we haven't specified explicitly -- via e.g. "`--allocation-pool start=<IP_address_pool_start>,end=<IP_address_pool_end>`" option -- what part of the address space is to be allocated dynamically. As a result all addresses in that range except the gateway address are automatically added to the pool available for VM instances.

Now we can check that this "network" and, respectively, "subnet" are available for tenant "endusers" (please note that we are using the "enduser1" Keystone credentials to do that):

```
######## As an end-user check which networks / subnets are available for provisioning

root@Folsom3:~# quantum --os_username enduser1  --os_password enduserpasswd1  --os_tenant_name endusers  --os_auth_url http://10.7.68.179:5001/v
+--------------------------------------+--------------+--------------------------------------+
| id                                   | name         | subnets                              |
+--------------------------------------+--------------+--------------------------------------+
| ff8a9831-3358-43e3-ae7f-bef03ff7bf6a | InternalNet1 | 099c2fc9-3992-4963-bfca-4bb1af8fdeea |
+--------------------------------------+--------------+--------------------------------------+

root@Folsom3:~# quantum --os_username enduser1  --os_password enduserpasswd1  --os_tenant_name endusers  --os_auth_url http://10.7.68.179:5001/v
+--------------------------------------+------+---------------+----------------------------------------------+
| id                                   | name | cidr          | allocation_pools                             |
+--------------------------------------+------+---------------+----------------------------------------------+
| 099c2fc9-3992-4963-bfca-4bb1af8fdeea |      | 172.16.1.0/24 | {"start": "172.16.1.2", "end": "172.16.1.254"} |
+--------------------------------------+------+---------------+----------------------------------------------+
```

Now we can do the same for tenant "services" -- in our setup the corresponding Keystone tenant_id is `0x8032be08210a424687ff3622338ffe23` (we omit the command output this time for brevity reasons):

```
######## Enter the Quantum CLI prompt using "admin" credentials

root@Folsom3:~# quantum --os_username quantumadmin --os_password quantumadminpasswd --os_tenant_name service --os_auth_url http://10.7.68.179:50
(quantum) net-create --tenant_id 8032be08210a424687ff3622338ffe23  SvcIntNet1 --provider:network_type gre  --provider:segmentation_id 999

# <output omitted>

(quantum) subnet-create --tenant_id 8032be08210a424687ff3622338ffe23 --ip_version 4  SvcIntNet1 169.254.1.0/24 --gateway 169.254.1.1

# <output omitted>
```

Similar to what we did for tenant "endusers" above, for tenant "services" we created network "SvcIntNet1" using GRE as a "provider" mechanism, using a "segmentation_id (i.e. GRE "tunnel-id") of 999 to identify the traffic on this network. This corresponds to a tunnel-id of `0x3e7` , as we will note below. The astute reader may note that the value for "tunnel-id" is within the range specified by the "tunnel_id_ranges" variable in the OVS agent configuration -- in our case between "1" and "1000".
We also associated a "subnet" with CIDR prefix of "`169.254.1.0/24`" with this network. Same observations as above apply (all addresses in that range except the gateway address are added to the DHCP pool for VM instances to use).

8.1 Starting VM instances -- different tenants on different networks

Once we have defined the networks and subnets for different tenants we can start VMs for those tenants. First is booting a VM for tenant "endusers":

```
# Check which image are available for provisioning

root@Folsom2:~# nova --os_username enduser1 --os_password enduserpasswd1 --os_tenant_name endusers  --os_auth_url http://10.7.68.179:5001/v2.0/
+--------------------------------------+---------------------+--------+--------+
| ID                                   | Name                | Status | Server |
+--------------------------------------+---------------------+--------+--------+
| 7789ebb5-14e8-4c44-a4ea-1bffdc52df51 | CirrOS-0.3.0-x86_64 | ACTIVE |        |
+--------------------------------------+---------------------+--------+--------+

# First, fix the default security group to allow us to ssh (tcp/22) and ping (ICMP) the instance

root@Folsom2:~# nova --os_username enduser1 --os_password enduserpasswd1 --os_tenant_name endusers  --os_auth_url http://10.7.68.179:5001/v2.0/
root@Folsom2:~# nova --os_username enduser1 --os_password enduserpasswd1 --os_tenant_name endusers  --os_auth_url http://10.7.68.179:5001/v2.0/
```

```
# Create some SSH keys and a key ring for VMs to use

root@Folsom2:~# ssh-keygen -q  -N "" -f Folsom-keypair
root@Folsom2:~# nova --os_username enduser1 --os_password enduserpasswd1 --os_tenant_name endusers  --os_auth_url http://10.7.68.179:5001/v2.0/

# Start a VM:
root@Folsom2:~# nova --os_username enduser1 --os_password enduserpasswd1 --os_tenant_name endusers  --os_auth_url http://10.7.68.179:5001/v2.0/
```

Please note that in those examples the Python Nova client was given the Keystone credentials for "enduser1" as CLI arguments. The reason for this was to make the context explicit but at the expense of brevity. In a practical environment we would be using shell aliases or ".rc" files.

Another observation is that "flavor 2" normally corresponds to "m1.small" instance type -- i.e. a VM with 1 VCPU, 2GB RAM and 30GB of local disk.

Next is to boot a VM for tenant "services" (used for "user-level" operations). The process is identical to the above except that the we would pass as Nova CLI arguments the Keystone credentials for user "novaadmin" like this:

```
nova --os_username novaadmin --os_password novaadminpasswd --os_tenant_name service  --os_auth_url http://10.7.68.179:5001/v2.0/ --os_region_nam
```

Since the process is identical to the above -- except the Keystone credentials, the name we may choose for the keypair and, obviously, the name of the VM instance -- we leave that as an exercise to the reader.

As a result we should have two running instances, one for each tenant / network / subnet, like this:

```
root@Folsom2:~# nova --os_username enduser1 --os_password enduserpasswd1 --os_tenant_name endusers  --os_auth_url http://10.7.68.179:5001/v2.0/
+--------------------------------------+-----------+--------+------------------------+
| ID                                   | Name      | Status | Networks               |
+--------------------------------------+-----------+--------+------------------------+
| ab702959-c904-4c9a-a52d-e2a4b07d0f7f | CirrOSv11 | ACTIVE | InternalNet1=172.16.1.3 |
+--------------------------------------+-----------+--------+------------------------+

root@Folsom2:~# nova --os_username novaadmin --os_password novaadminpasswd --os_tenant_name service  --os_auth_url http://10.7.68.179:5001/v2.0/
+--------------------------------------+-----------+--------+------------------------+
| ID                                   | Name      | Status | Networks               |
+--------------------------------------+-----------+--------+------------------------+
| 0feab3f7-6eda-4691-a023-249cf471546b | CirrOSv12 | ACTIVE | SvcIntNet1=169.254.1.3 |
+--------------------------------------+-----------+--------+------------------------+
```

As expected, each of the two instances (in our case "CirrOSv11" for tenant "endusers" and, respectively, "CirrOSv12" for tenant "services") was dynamically assiged an IP address from the network / subnet assigned to those tenants, respectively.

8.2 Resulting working configuration -- Quantum

At this point we can take a closer look at the objects that are instantiated in Quantum as part of the above process:

```
######## Enter the Quantum CLI prompt using "admin" credentials
root@Folsom3:~# quantum --os_username quantumadmin --os_password quantumadminpasswd --os_tenant_name service --os_auth_url http://10.7.68.179:50

######## List the networks that are created

(quantum) net-list
+--------------------------------------+--------------+--------------------------------------+
| id                                   | name         | subnets                              |
+--------------------------------------+--------------+--------------------------------------+
| e2fcc64c-ec17-4658-bef2-4685a3f06dfe | SvcIntNet1   | 9e8a5cdb-eaa6-4a4b-a8f0-5e372703001b |
| ff8a9831-3358-43e3-ae7f-bef03ff7bf6a | InternalNet1 | 099c2fc9-3992-4963-bfca-4bb1af8fdeea |
+--------------------------------------+--------------+--------------------------------------+

######## Each of them is defined for a different tenant and "segmentation_id" (GRE "tunnel-id")

(quantum) net-show ff8a9831-3358-43e3-ae7f-bef03ff7bf6a
+---------------------------+--------------------------------------+
| Field                     | Value                                |
+---------------------------+--------------------------------------+
| admin_state_up            | True                                 |
| id                        | ff8a9831-3358-43e3-ae7f-bef03ff7bf6a |
| name                      | InternalNet1                         |
| provider:network_type     | gre                                  |
| provider:physical_network |                                      |
| provider:segmentation_id  | 1                                    |
| router:external           | False                                |
| shared                    | False                                |
| status                    | ACTIVE                               |
| subnets                   | 099c2fc9-3992-4963-bfca-4bb1af8fdeea |
| tenant_id                 | 9be459eedc674635bbef86a47ac8896d     |
+---------------------------+--------------------------------------+

(quantum) net-show e2fcc64c-ec17-4658-bef2-4685a3f06dfe
+---------------------------+--------------------------------------+
| Field                     | Value                                |
+---------------------------+--------------------------------------+
| admin_state_up            | True                                 |
| id                        | e2fcc64c-ec17-4658-bef2-4685a3f06dfe |
| name                      | SvcIntNet1                           |
| provider:network_type     | gre                                  |
| provider:physical_network |                                      |
| provider:segmentation_id  | 999                                  |
| router:external           | False                                |
| shared                    | False                                |
| status                    | ACTIVE                               |
| subnets                   | 9e8a5cdb-eaa6-4a4b-a8f0-5e372703001b |
| tenant_id                 | 8032be08210a424687ff3622338ffe23     |
+---------------------------+--------------------------------------+


######## The subnets we created

(quantum) subnet-list
+--------------------------------------+------+----------------+--------------------------------------------------+
| id                                   | name | cidr           | allocation_pools                                 |
+--------------------------------------+------+----------------+--------------------------------------------------+
| 099c2fc9-3992-4963-bfca-4bb1af8fdeea |      | 172.16.1.0/24  | {"start": "172.16.1.2", "end": "172.16.1.254"}   |
| 9e8a5cdb-eaa6-4a4b-a8f0-5e372703001b |      | 169.254.1.0/24 | {"start": "169.254.1.2", "end": "169.254.1.254"} |
+--------------------------------------+------+----------------+--------------------------------------------------+

######## Each of those two subnets belong to the a different Tenant / Network combination

(quantum) subnet-show 099c2fc9-3992-4963-bfca-4bb1af8fdeea
+------------------+------------------------------------------------+
| Field            | Value                                          |
+------------------+------------------------------------------------+
| allocation_pools | {"start": "172.16.1.2", "end": "172.16.1.254"} |
| cidr             | 172.16.1.0/24                                  |
| dns_nameservers  |                                                |
| enable_dhcp      | True                                           |
| gateway_ip       | 172.16.1.1                                     |
| host_routes      |                                                |
| id               | 099c2fc9-3992-4963-bfca-4bb1af8fdeea           |
| ip_version       | 4                                              |
| name             |                                                |
| network_id       | ff8a9831-3358-43e3-ae7f-bef03ff7bf6a           |
| tenant_id        | 9be459eedc674635bbef86a47ac8896d               |
+------------------+------------------------------------------------+

(quantum) subnet-show 9e8a5cdb-eaa6-4a4b-a8f0-5e372703001b
+------------------+--------------------------------------------------+
| Field            | Value                                            |
+------------------+--------------------------------------------------+
| allocation_pools | {"start": "169.254.1.2", "end": "169.254.1.254"} |
| cidr             | 169.254.1.0/24                                   |
| dns_nameservers  |                                                  |
| enable_dhcp      | True                                             |
| gateway_ip       | 169.254.1.1                                      |
| host_routes      |                                                  |
| id               | 9e8a5cdb-eaa6-4a4b-a8f0-5e372703001b             |
| ip_version       | 4                                                |
| name             |                                                  |
| network_id       | e2fcc64c-ec17-4658-bef2-4685a3f06dfe             |
| tenant_id        | 8032be08210a424687ff3622338ffe23                 |
+------------------+--------------------------------------------------+


######## The active ports after we booted one VN instance for each of the two Tenant/Network/Subnet combos

(quantum) port-list
+--------------------------------------+------+-------------------+------------------------------------------------------------------------------------------
```

```
| id                                   | name | mac_address       | fixed_ips                                                                           
+--------------------------------------+------+-------------------+-------------------------------------------------------------------------------------
| 0e23ab9b-d60c-4a10-8958-b0055def6dde |      | fa:16:3e:16:10:3a | {"subnet_id": "099c2fc9-3992-4963-bfca-4bb1af8fdeea", "ip_address": "172.16.
| 796f8a41-7209-417e-a0a2-fb85cd45f04b |      | fa:16:3e:41:c0:24 | {"subnet_id": "9e8a5cdb-eaa6-4a4b-a8f0-5e372703001b", "ip_address": "169.254
| 906fc936-96ed-4d77-a688-75d93c650a80 |      | fa:16:3e:e8:bd:42 | {"subnet_id": "099c2fc9-3992-4963-bfca-4bb1af8fdeea", "ip_address": "172.16.
| 921cedd1-311a-4974-93aa-4e773aa19f20 |      | fa:16:3e:dc:c5:23 | {"subnet_id": "9e8a5cdb-eaa6-4a4b-a8f0-5e372703001b", "ip_address": "169.254
+--------------------------------------+------+-------------------+-------------------------------------------------------------------------------------
```

With regards to the last command -- the listing of "ACTIVE" ports -- the astute reader may note two things:

- When we created the VM instances above they were both assigned an IP address of "`...1.3`" from their respective address space, even if the allocation pool started with "`...1.2`"
- The corresponding "`...1.2`" IP address from those address spaces is listed as an active port associated with it.

A cursory inspection on the Network reveals that these addresses are assigned to the Quantum DHCP agent handling the IP address allocation for those IP address spaces (i.e. subnets) :

```
root@Folsom3:~# ps axuw | grep -i dnsmasq
root      8510  0.0  0.0   8104   892 pts/0    S+   12:50   0:00 grep --color=auto -i dnsmasq
root     12122  0.0  1.1  90460 24232 ?        S    Feb18   0:03 /usr/bin/python /usr/local/bin/quantum-ns-metadata-proxy --pid_file=/var/run/dn
nobody   24386  0.0  0.0  27540   944 ?        S    Jan24   0:09 dnsmasq --no-hosts --no-resolv --strict-order --bind-interfaces --interface=tap
root     24387  0.0  0.0  27512   452 ?        S    Jan24   0:10 dnsmasq --no-hosts --no-resolv --strict-order --bind-interfaces --interface=tap
nobody   24407  0.0  0.0  27540   940 ?        S    Jan24   0:09 dnsmasq --no-hosts --no-resolv --strict-order --bind-interfaces --interface=tap
root     24408  0.0  0.0  27512   448 ?        S    Jan24   0:11 dnsmasq --no-hosts --no-resolv --strict-order --bind-interfaces --interface=tap
root@Folsom3:~#
```

A few observations are in order:

- For each of the two adddress spaces (subnets) we have two "dnsmasq" processes running. This is in order to provide some sort of redundancy at the DHCP server level.
- The IP address allocation policy is "static".
- For each of these two ranges the two DHCP servers for that space share the same network interface. In our case those interfaces are "`tap906fc936-96`" for the `172.16.1.0/24` subnet and, respectively, "`tap796f8a41-72`" for the `169.254.1.0/24` subnet. Like their names imply, these "tap" network interfaces are created dynamically by the Quantum DHCP agent.

Another very important observation is that by default the Quantum DHCP agent is using Linux network namespaces -- as we noted in previous section when we configured the DHCP agent. The reason is to allow different tenants use same or overlapping address spaces but still segregate between them.

As a result the two network interfaces above are not visibile in the global network namespace on the Network server. Instead we need to connect to their corresponding namespace (one network namespace per subnet). Please note we need to do that even if the subnets are distinct / non-overlapping (as in our case):

```
######## List the network namespaces on the Network server
root@Folsom3:~# ip netns list
qdhcp-e2fcc64c-ec17-4658-bef2-4685a3f06dfe
qdhcp-ff8a9831-3358-43e3-ae7f-bef03ff7bf6a

######## When executing commands we need to do that in the right context (i.e. network namespace) e.g. "ip netns exec <netwwork_namespace> <OS_c

root@server-1355824565-az-2-region-a-geo-1:~# ip netns exec qdhcp-e2fcc64c-ec17-4658-bef2-4685a3f06dfe ifconfig
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:16436  Metric:1
          RX packets:53 errors:0 dropped:0 overruns:0 frame:0
          TX packets:53 errors:0 dropped:0 overruns:0 carrier:0
          collisions:0 txqueuelen:0
          RX bytes:5584 (5.5 KB)  TX bytes:5584 (5.5 KB)

tap796f8a41-72 Link encap:Ethernet  HWaddr fa:16:3e:41:c0:24
          inet addr:169.254.1.2  Bcast:169.254.1.255  Mask:255.255.255.0
          inet6 addr: fe80::f816:3eff:fe41:c024/64 Scope:Link
          UP BROADCAST RUNNING MULTICAST  MTU:1500  Metric:1
          RX packets:139896 errors:0 dropped:0 overruns:0 frame:0
          TX packets:99760 errors:0 dropped:0 overruns:0 carrier:0
          collisions:0 txqueuelen:0
          RX bytes:32031451 (32.0 MB)  TX bytes:18101097 (18.1 MB)

root@server-1355824565-az-2-region-a-geo-1:~# ip netns exec qdhcp-ff8a9831-3358-43e3-ae7f-bef03ff7bf6a ifconfig
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:16436  Metric:1
          RX packets:18 errors:0 dropped:0 overruns:0 frame:0
          TX packets:18 errors:0 dropped:0 overruns:0 carrier:0
          collisions:0 txqueuelen:0
          RX bytes:1776 (1.7 KB)  TX bytes:1776 (1.7 KB)

tap906fc936-96 Link encap:Ethernet  HWaddr fa:16:3e:e8:bd:42
          inet addr:172.16.1.2  Bcast:172.16.1.255  Mask:255.255.255.0
          inet6 addr: fe80::f816:3eff:fee8:bd42/64 Scope:Link
          UP BROADCAST RUNNING MULTICAST  MTU:1500  Metric:1
          RX packets:129024 errors:0 dropped:0 overruns:0 frame:0
          TX packets:92606 errors:0 dropped:0 overruns:0 carrier:0
          collisions:0 txqueuelen:0
          RX bytes:29236423 (29.2 MB)  TX bytes:16686585 (16.6 MB)
```

From the Network server it is only from the corresponding network namespace that we can reach the VM instances on the Compute node:

```
######## We cannot reach any of the VM instances from the global network namespace

root@Folsom3:~# ping -c 3 172.16.1.3
PING 172.16.1.3 (172.16.1.3) 56(84) bytes of data.
From 172.16.1.1 icmp_seq=1 Destination Net Unreachable
From 172.16.1.1 icmp_seq=2 Destination Net Unreachable
From 172.16.1.1 icmp_seq=3 Destination Net Unreachable

--- 172.16.1.3 ping statistics ---
3 packets transmitted, 0 received, +3 errors, 100% packet loss, time 2016ms

root@Folsom3:~# ping -c 3 169.254.1.3
PING 169.254.1.3 (169.254.1.3) 56(84) bytes of data.
From 205.129.16.13 icmp_seq=1 Destination Net Unreachable
From 205.129.16.13 icmp_seq=2 Destination Net Unreachable
From 205.129.16.13 icmp_seq=3 Destination Net Unreachable

--- 169.254.1.3 ping statistics ---
3 packets transmitted, 0 received, +3 errors, 100% packet loss, time 2002ms

######## But we can do that from the corresponding namespace

root@Folsom3:~# ip netns exec qdhcp-ff8a9831-3358-43e3-ae7f-bef03ff7bf6a ping -c 3 172.16.1.3
PING 172.16.1.3 (172.16.1.3) 56(84) bytes of data.
64 bytes from 172.16.1.3: icmp_req=1 ttl=64 time=5.93 ms
64 bytes from 172.16.1.3: icmp_req=2 ttl=64 time=3.44 ms
64 bytes from 172.16.1.3: icmp_req=3 ttl=64 time=3.28 ms

--- 172.16.1.3 ping statistics ---
3 packets transmitted, 3 received, 0% packet loss, time 2002ms
rtt min/avg/max/mdev = 3.287/4.223/5.939/1.216 ms

root@Folsom3:~# ip netns exec qdhcp-e2fcc64c-ec17-4658-bef2-4685a3f06dfe ping -c 3 169.254.1.3
PING 169.254.1.3 (169.254.1.3) 56(84) bytes of data.
64 bytes from 169.254.1.3: icmp_req=1 ttl=64 time=5.99 ms
64 bytes from 169.254.1.3: icmp_req=2 ttl=64 time=3.05 ms
64 bytes from 169.254.1.3: icmp_req=3 ttl=64 time=3.48 ms

--- 169.254.1.3 ping statistics ---
3 packets transmitted, 3 received, 0% packet loss, time 2002ms
rtt min/avg/max/mdev = 3.053/4.175/5.991/1.295 ms
```

Next we will be looking at the corresponding OVS configuration.

8.3 Resulting working configuration -- Open vSwitch

As we detailed above when we described the Quantum architecture the objects and constructs defined in Quantum are actually implemented by the back-end plugin -- in our case Open vSwitch. In this section we will illustrate how the setup we created above is instantiated at the OVS level.

First, let's take a look at the resulting OVS configuration on the Network node, "Folsom 3":

```
root@Folsom3:~# ovs-vsctl show
e26dca84-dbfc-430a-b54f-2ee076a20b32
    Bridge br-tun
        Port "gre-1"
            Interface "gre-1"
                type: gre
                options: {in_key=flow, out_key=flow, remote_ip="192.168.1.253"}
        Port br-tun
            Interface br-tun
                type: internal
        Port patch-int
            Interface patch-int
                type: patch
                options: {peer=patch-tun}
    Bridge br-ex
        Port "eth0"
            Interface "eth0"
        Port br-ex
            Interface br-ex
                type: internal
    Bridge br-int
        Port patch-tun
            Interface patch-tun
                type: patch
                options: {peer=patch-int}
        Port "tap796f8a41-72"
            tag: 1
            Interface "tap796f8a41-72"
                type: internal
        Port br-int
            Interface br-int
                type: internal
        Port "tap906fc936-96"
            tag: 2
            Interface "tap906fc936-96"
                type: internal
root@Folsom3:~#
```

From bottom up (omitting bridge internal interfaces):

- As described in the OVS agent configuration file "/etc/quantum/plugins/openvswitch/ovs_quantum_plugin.ini", "br-int" is the so called "integration bridge". This bridge serves a "patch panel" where all VM instances will be connected (as we will see below for the Compute node) and, in the case of the Network node here, where the Quantum-managed DHCP servers are connected to. As such on the Network node here we readily recognize the network interfaces "`tap906fc936-96`" as the shared network interface of the two DHCP servers for `172.16.1.0/24` subnet and, respectively, "`tap796f8a41-72`" for the two DHCP servers for the `169.254.1.0/24` subnet.

  As apart of the integration bridge we see listed patch port "patch-tun". Specified in the OVS agent configuration file are two patchs ports -- "patch-tun" and, respectively, "patch-int". As their name imply these ports serve as a connection between the integration bridge "br-int" and the tunnel bridge "br-tun" below.

- The external bridge "br-ex" was defined in the L3 agent configuration file "/etc/quantum/l3_agent.ini" and is used for connecting to external network resoruces. This is why the server network interface "eth0" was manually added to this bridge (we left this step outside the text). For more details, see **Limitations** section below.

- The tunnel bridge "br-tun" is where the GRE tunnels between the Network node and the Compute node are terminated. As we see here this GRE termination port "gre-1" lists as peer the IP address of the other end of the GRE tunnel -- in our case the IP address of the Compute node on the Data network.

Let's take a closer look at the GRE-encapsulated network traffic between Network and Compute node. First, from the Network node perspective:

```
######## Look at the in-kernel datapaths and identify the port numbers in the different bridges

root@Folsom3:~# ovs-dpctl show
system@br-tun:
        lookups: hit:727 missed:18710 lost:0
        flows: 3
        port 0: br-tun (internal)
        port 2: patch-int (patch: peer=patch-tun)
        port 4: gre-1 (gre: key=flow, remote_ip=192.168.1.253)
system@br-int:
        lookups: hit:2874 missed:472748 lost:0
        flows: 3
        port 0: br-int (internal)
        port 13: tap906fc936-96 (internal)
        port 14: tap796f8a41-72 (internal)
        port 24: patch-tun (patch: peer=patch-int)
system@br-ex:
        lookups: hit:3718320 missed:69301732 lost:0
        flows: 78
        port 0: br-ex (internal)
        port 1: eth0

####### Look at the forwarding rules in the tunneling bridge
```

```
root@Folsom3:~# ovs-ofctl dump-flows br-tun
NXST_FLOW reply (xid=0x4):
 cookie=0x0, duration=168969.459s, table=0, n_packets=3628, n_bytes=705412, idle_age=2, hard_age=65534, priority=3,tun_id=0x3e7,dl_dst=fa:16:3e:
 cookie=0x0, duration=168968.568s, table=0, n_packets=4129, n_bytes=794611, idle_age=24, hard_age=65534, priority=3,tun_id=0x1,dl_dst=fa:16:3e:e
 cookie=0x0, duration=168969.491s, table=0, n_packets=1656, n_bytes=508742, idle_age=2, hard_age=65534, priority=3,tun_id=0x3e7,dl_dst=01:00:00:
 cookie=0x0, duration=168969.024s, table=0, n_packets=1850, n_bytes=569134, idle_age=29, hard_age=65534, priority=3,tun_id=0x1,dl_dst=01:00:00:0
 cookie=0x0, duration=168970.962s, table=0, n_packets=9, n_bytes=1470, idle_age=65534, hard_age=65534, priority=1 actions=drop
 cookie=0x0, duration=168969.522s, table=0, n_packets=3663, n_bytes=665006, idle_age=2, hard_age=65534, priority=4,in_port=2,dl_vlan=1 actions=s
 cookie=0x0, duration=168969.055s, table=0, n_packets=4334, n_bytes=771331, idle_age=24, hard_age=65534, priority=4,in_port=2,dl_vlan=2 actions=
```

One thing to note there is that the first rule for "br-tun" instructs that any traffic with GRE tunnel-id of "999" (hexadecimal `0x3e7`) destined to MAC address `fa:16:3e:41:c0:24` (which, as can be seen above, is the MAC address of "`tap796f8a41-72`" interface of the DHCP servers for the `169.254.1.0/24` subnet) is first to be VLAN tagged with VLAN Id of "1" before any further processing. Correspondingly, the third rule instructs that any GRE traffic of a tunnel-id of `0x3e7` that is Ethernet multicast / broadcast -- in OVS the **dl_dst=01:00:00:00:00:00/01:00:00:00:00:00** rule matches all multicast and broadcast Ethernet packets -- is to be marked with VLAN Id of "1" before being sent to port nr "2" of "br-tun". Which, as we can seen from the output of "ovs-dpctl", is the "patch-int" port. In other words the traffic should be marked with VLAN Id of "1" and then send directly to "br-int" with no further processing by the rest of the forwrding rules.

In the reverse direction, the penultimate rule in the forwarding set instructs that any traffic arriving from "br-int" via patch port "patch-int" (port Id of "2" in "br-tun") that is tagged with VLAN Id of "1" is to sent as GRE traffic with tunnel-id of `0x3e7`.

To sum it up, any traffic that is to be sent to -- or is being received from -- GRE tunnel-id of `0x3e7` (i.e. traffic on the `169.254.1.0/24` subnet) is to be internally tagged with VLAN Id of "1".

The remainder of the rules in the forwarding set for "br-tun" denote the traffic forwarding actions for the other subnet in our setup -- namely that all traffic to/from GRE tunnel-id of `0x1` (i.e. traffic on the `172.16.1.0/24` subnet) is to be internally tagged with VLAN Id of "2": The second rule mandates that traffic with GRE tunnel-Id of "1" (i.e. `0x1`) destined to MAC address `fa:16:3e:e8:bd:42` (the MAC address of "`tap906fc936-96`" interface, the inteface for DHCP servers on `172.16.1.0/24`) is to be first VLAN tagged with VLAN Id of "2". Same action is taking for any multicast/broadcast Ethernet traffic that is arriving with a GRE tunnel-Id of "1" (4th rule) before sending it to "br-int". And, finally, for the traffic in the reverse direction the last rule in the forwarding set instructs that any traffic marked with VLAN Id of "2" is to be sent as GRE traffic with GRE tunnel-id of "1".

At this point the astute reader may point out that this could have been already noted in the output of the "ovs-vsctl show" command above: Traffic to/from the"`tap796f8a41-72`" interface is to tagged with a VLAN Id of "1" whereas traffic to/from "`tap906fc936-96`" is tagged with VLAN Id of "2".

An important observation here is in order: This use of VLAN tagging is to seggregate the different types of traffic as traffic is forwarded between "br-tun" and "br-int" internally in the Network node. This is independent of any configuration setting and it is a Quantum specific implementation detail.

Let's take a look at the corresponding OVS configuration on the Compute node:

```
######## Look at the in-kernel datapaths and identify the port numbers in the different bridges

root@Folsom2:~# ovs-vsctl show
8aa4bbf9-e536-4c23-aaa5-5d81ba16073b
    Bridge br-tun
        Port br-tun
            Interface br-tun
                type: internal
        Port patch-int
            Interface patch-int
                type: patch
                options: {peer=patch-tun}
        Port "gre-2"
            Interface "gre-2"
                type: gre
                options: {in_key=flow, out_key=flow, remote_ip="192.168.1.254"}
    Bridge br-int
        Port patch-tun
            Interface patch-tun
                type: patch
                options: {peer=patch-int}
        Port "tap921cedd1-31"
            tag: 7
            Interface "tap921cedd1-31"
        Port br-int
            Interface br-int
                type: internal
        Port "tap0e23ab9b-d6"
            tag: 6
            Interface "tap0e23ab9b-d6"
```

A few things to note briefly:

- In the tunnel bridge "br-tun" the "gre-2" port is the GRE tunnel termination end-point on the Compute node.
- In the integration bridge "br-int" we have two interfaces: "`tap0e23ab9b-d6`" interface and "`tap921cedd1-31`" . These correspond to our two VMs we instantiated above.

One observation is in order. The details of those two interfaces (name, MAC address etc) that we can list on the Compute node using the traditional OS tools are from the server point of view. These do not necessarily match the details of those interfaces from the VM instance perspective:

```
root@Folsom2:~# ovs-dpctl show
system@br-int:
        lookups: hit:13427 missed:481789 lost:0
        flows: 0
        port 0: br-int (internal)
        port 36: patch-tun (patch: peer=patch-int)
        port 40: tap0e23ab9b-d6
        port 41: tap921cedd1-31
system@br-tun:
        lookups: hit:11305 missed:362807 lost:0
        flows: 0
        port 0: br-tun (internal)
        port 2: patch-int (patch: peer=patch-tun)
        port 4: gre-2 (gre: key=flow, remote_ip=192.168.1.254)

####### Look at the forwarding configuration in the tunneling bridge

root@server-1355685201-az-2-region-a-geo-1:~# ovs-ofctl dump-flows br-tun
NXST_FLOW reply (xid=0x4):
 cookie=0x0, duration=114832.491s, table=0, n_packets=4549, n_bytes=825519, idle_age=35, hard_age=65534, priority=3,tun_id=0x3e7,dl_dst=fa:16:3e
 cookie=0x0, duration=114978.637s, table=0, n_packets=4488, n_bytes=813368, idle_age=2, hard_age=65534, priority=3,tun_id=0x1,dl_dst=fa:16:3e:16
 cookie=0x0, duration=114832.515s, table=0, n_packets=0, n_bytes=0, idle_age=65534, hard_age=65534, priority=3,tun_id=0x3e7,dl_dst=01:00:00:00:0
 cookie=0x0, duration=114978.66s, table=0, n_packets=0, n_bytes=0, idle_age=65534, hard_age=65534, priority=3,tun_id=0x1,dl_dst=01:00:00:00:00:0
 cookie=0x0, duration=114832.536s, table=0, n_packets=6522, n_bytes=1496811, idle_age=35, hard_age=65534, priority=4,in_port=2,dl_vlan=7 actions
 cookie=0x0, duration=114978.685s, table=0, n_packets=6494, n_bytes=1488044, idle_age=2, hard_age=65534, priority=4,in_port=2,dl_vlan=6 actions=
 cookie=0x0, duration=2089910.837s, table=0, n_packets=1, n_bytes=46, idle_age=65534, hard_age=65534, priority=1 actions=drop
```

Taking a closer look at the forwarding rules in "br-tun" we see they mirror the ones on the Network node -- albeit with some differences worth mentioning:

The first rule in the set instructs that any traffic arriving with GRE tunnel-id of `0x3e7` (decimal 999) and destined to MAC address `fa:16:3e:dc:c5:23` is to be marked with VLAN Id of "7". As we can see from the Quantum CLI "port-list" command above, this MAC address corresponds to IP address `169.254.1.3` , which is the IP address of the "CirrOSv12" -- the VM we created for tenant "services". Correspondingly, the 3rd rule in the set marks in the same way any Ethernet broadcast/multicast traffic arriving with that GRE tunnel-id. Lastly, for the reverse traffic direction the 5th rule in the set instructs that any traffic received from the integration bridge via the patch port "patch-int" that is marked with VLAN Id of "7" should be encapsulated as GRE traffic with tunnel-id of `0x3e7`.

We leave interpreting the forwarding rules for the traffic to/from the VM for tenant "endusers" (MAC address `fa:16:3e:16:10:3a` , IP address `172.16.1.3`) as an exercise to the reader.

From a server perspective we cannot directly tell which network interface in the integration bridge corresponds to which tenant VM. However, from the output of "ovs-vsctl show" above, correlated with this information we can deduce that interface "`tap921cedd1-31`" (traffic tagged with VLAN Id of "7") corresponds to the VM for tenant "services" (MAC address `fa:16:3e:dc:c5:23` , IP address `169.254.1.3`) whereas interface "`tap0e23ab9b-d6`" (tagged with VLAN Id of "6") corresponds to the VM for tenant "endusers" (MAC address `fa:16:3e:16:10:3a` , IP address `172.16.1.3`).

Like we noted for the Network node above, this use of VLAN tagging is to seggrate the traffic between the tunneling bridge and the integration bridge and it is a Quantum implementation detail.

All those configuration details can be also observed by analyzing the traffic on the Data network between the Network and Compute node. For example when we "ping" (i.e. ICMP traffic) the VM for tenant "services" from the Network node:

```
######## Make sure we do this from the right network space

root@Folsom3:~# ip netns exec qdhcp-ff8a9831-3358-43e3-ae7f-bef03ff7bf6a ping -c 1 172.16.1.3
PING 172.16.1.3 (172.16.1.3) 56(84) bytes of data.
64 bytes from 172.16.1.3: icmp_req=1 ttl=64 time=5.71 ms

--- 172.16.1.3 ping statistics ---
1 packets transmitted, 1 received, 0% packet loss, time 0ms
rtt min/avg/max/mdev = 5.716/5.716/5.716/0.000 ms
```

The corresponding traffic trace on the Data network is:

```
17:04:11.595393 ip: (tos 0x0, ttl 64, id 38209, offset 0, flags [DF], proto GRE (47), length 130)
    192.168.1.254 > 192.168.1.253: GREv0, Flags [key present], key=0x1, proto TEB (0x6558), length 110
```

```
        fa:16:3e:e8:bd:42 > fa:16:3e:16:10:3a, ethertype 802.1Q (0x8100), length 102: vlan 2, p 0, ethertype IPv4, (tos 0x0, ttl 64, id 0, offse
    172.16.1.2 > 172.16.1.3: ICMP echo request, id 14613, seq 1, length 64
17:04:11.597439 ip: (tos 0x0, ttl 64, id 50389, offset 0, flags [DF], proto GRE (47), length 130)
    192.168.1.253 > 192.168.1.254: GREv0, Flags [key present], key=0x1, proto TEB (0x6558), length 110
        fa:16:3e:16:10:3a > fa:16:3e:e8:bd:42, ethertype 802.1Q (0x8100), length 102: vlan 6, p 0, ethertype IPv4, (tos 0x0, ttl 64, id 4859, of
    172.16.1.3 > 172.16.1.2: ICMP echo reply, id 14613, seq 1, length 64
```

A few things to note. First, on the "ICMP echo request packet":

- As expected the traffic is GRE encapsulated, with a GRE tunnel-id of `0x1`. In the outer IP header the source / destination IP addresses are those of the GRE tunnel endpoints on the Network node (`192.168.1.254`) and, respectively, Compute node (`192.168.1.253`).
- Since we do this from the corresponding network namespace the traffic is sent via the network interface of the DHCP servers on that subnet (i.e. the "`tap906fc936-96`" interface with MAC address `fa:16:3e:e8:bd:42` / IP address `172.16.1.2`).
- Most notablly, the packet is VLAN tagged with ta VLAN Id of "2". While we noted above that the use of VLAN tagging is internal in the Network node between "br-int" and "br-tun" this is still visible on the Data Network.

  While benign, having this Network node - internal VLAN Id being "spilled" on the Data network may be considered as a "bug".

For the return packet the same observations hold: Traffic is GRE-encapsulated, with the outer IP header denoting the GRE tunnel endpoints on the Compute and, respectively, Network node. Also, the same observation applies: One may be well entitled to consider as an "implementation bug" exposing on the Data Network the VLAN Id used internally on the Compute Node for tagging that traffic (in this case the VLAN Id of "6").

Let's do the same for the VM we created for the "endusers" tenant:

```
######## Make sure to do this from the right network space on the Network node

root@Folsom3:~# ip netns exec qdhcp-e2fcc64c-ec17-4658-bef2-4685a3f06dfe ping -c 1 169.254.1.3
PING 169.254.1.3 (169.254.1.3) 56(84) bytes of data.
64 bytes from 169.254.1.3: icmp_req=1 ttl=64 time=5.93 ms

--- 169.254.1.3 ping statistics ---
1 packets transmitted, 1 received, 0% packet loss, time 0ms
rtt min/avg/max/mdev = 5.932/5.932/5.932/0.000 ms
```

And the corresponding packet trace on the Data Network:

```
11:32:14.866202 ip: (tos 0x0, ttl 64, id 56642, offset 0, flags [DF], proto GRE (47), length 130)
    192.168.1.254 > 192.168.1.253: GREv0, Flags [key present], key=0x3e7, proto TEB (0x6558), length 110
        fa:16:3e:41:c0:24 > fa:16:3e:dc:c5:23, ethertype 802.1Q (0x8100), length 102: vlan 1, p 0, ethertype IPv4, (tos 0x0, ttl 64, id 0, offse
    169.254.1.2 > 169.254.1.3: ICMP echo request, id 15208, seq 1, length 64
11:32:14.868517 ip: (tos 0x0, ttl 64, id 11158, offset 0, flags [DF], proto GRE (47), length 130)
    192.168.1.253 > 192.168.1.254: GREv0, Flags [key present], key=0x3e7, proto TEB (0x6558), length 110
        fa:16:3e:dc:c5:23 > fa:16:3e:41:c0:24, ethertype 802.1Q (0x8100), length 102: vlan 7, p 0, ethertype IPv4, (tos 0x0, ttl 64, id 46854, o
    169.254.1.3 > 169.254.1.2: ICMP echo reply, id 15208, seq 1, length 64
```

We leave interpreting this packet trace and correlating it with the values above as an exercise to the reader. The same observations apply.

8.4. Connectivity to external resources -- L3 agent

While some L3 functionality can be already implemented in Quantum -- e.g. with per-tenant topologies -- it can be argued that at the time of this writing this is one of the less mature functionalities in Quantum. As it is firmly outside the scope of this text to argue what functionality should (or should not) be implemented as part of the Quantum framework, here we only illustrate the functionality at Layer 3 (and above) "as is" today. Please see here (or even here) for the latest status, e.g. this blueprint as an example of recent work on this topic or this blueprint for proposal for future improvements.

Like we noted already connectivity to Quantum / OpensStack external network resources is currently provided by the Quantum L3 agent. As described in Quantum documenation the basic L3 worflow is as follows:

```
####### Enter the Quantum CLI prompt using "admin" credentials
root@Folsom3:~# quantum --os_username quantumadmin --os_password quantumadminpasswd --os_tenant_name service --os_auth_url http://10.7.68.179:50

######## Create an external network called "HPCS_private"
```

```
(quantum) net-create HPCS_private --router:external=True
+---------------------------+--------------------------------------+
| Field                     | Value                                |
+---------------------------+--------------------------------------+
| admin_state_up            | True                                 |
| id                        | 693dbf84-e6dc-4cc1-8fe4-162674416ec9 |
| name                      | HPCS_private                         |
| provider:network_type     | gre                                  |
| provider:physical_network |                                      |
| provider:segmentation_id  | 2                                    |
| router:external           | True                                 |
| shared                    | False                                |
| status                    | ACTIVE                               |
| subnets                   |                                      |
| tenant_id                 | 8032be08210a424687ff3622338ffe23     |
+---------------------------+--------------------------------------+

######## For this network create a subnet -- 10.6.0.0/15 , gateway 10.6.0.1

(quantum) subnet-create --ip_version 4 --gateway 10.6.0.1  693dbf84-e6dc-4cc1-8fe4-162674416ec9 10.6.0.0/15 --enable_dhcp=False
Created a new subnet:
+------------------+--------------------------------------------+
| Field            | Value                                      |
+------------------+--------------------------------------------+
| allocation_pools | {"start": "10.6.0.2", "end": "10.7.255.254"} |
| cidr             | 10.6.0.0/15                                |
| dns_nameservers  |                                            |
| enable_dhcp      | False                                      |
| gateway_ip       | 10.6.0.1                                   |
| host_routes      |                                            |
| id               | 7f19e76e-d3a3-4800-8e98-8a9e2f272180       |
| ip_version       | 4                                          |
| name             |                                            |
| network_id       | 693dbf84-e6dc-4cc1-8fe4-162674416ec9       |
| tenant_id        | 8032be08210a424687ff3622338ffe23           |
+------------------+--------------------------------------------+

######## Create a router called "InternalNet1_GW" for tenant "endusers"

(quantum) router-create --tenant_id 9be459eedc674635bbef86a47ac8896d InternalNet1_GW
Created a new router:
+-----------------------+--------------------------------------+
| Field                 | Value                                |
+-----------------------+--------------------------------------+
| admin_state_up        | True                                 |
| external_gateway_info |                                      |
| id                    | c7073943-dee9-4e23-ac3c-447d86a82959 |
| name                  | InternalNet1_GW                      |
| status                | ACTIVE                               |
| tenant_id             | 9be459eedc674635bbef86a47ac8896d     |
+-----------------------+--------------------------------------+

########  For this router add an interface on subnet 172.16.1.0/24. This interface will automatically be assgined the default GW address on that

(quantum) router-interface-add c7073943-dee9-4e23-ac3c-447d86a82959 099c2fc9-3992-4963-bfca-4bb1af8fdeea
Added interface to router c7073943-dee9-4e23-ac3c-447d86a82959

######## Make this router gateway to the "HPCS_private" network above

(quantum) router-gateway-set c7073943-dee9-4e23-ac3c-447d86a82959 693dbf84-e6dc-4cc1-8fe4-162674416ec9
Set gateway for router c7073943-dee9-4e23-ac3c-447d86a82959
```

As a result of those actions the "InternalNet1_GW" router should act as SNAT gateway to the external "HPCS_private network (i.e. `10.6.0.0/15` subnet) for all the VMs belonging to tenant "endusers" on the `172.16.1.0/24` subnet.

One thing to note is that when we configured the Quantum L3 agent in previous section we specified the use of networking namespaces ("use_namespaces = True"). As such when we created this router on the Network node a new networking namespace was instantiated:

```
root@Folsom3:~# ip netns
qrouter-c7073943-dee9-4e23-ac3c-447d86a82959
qdhcp-e2fcc64c-ec17-4658-bef2-4685a3f06dfe
qdhcp-ff8a9831-3358-43e3-ae7f-bef03ff7bf6a

######## Look at the network interfaces created for this router (in the appropriate namespace)
root@Folsom3:~#  ip netns exec qrouter-c7073943-dee9-4e23-ac3c-447d86a82959 ifconfig
```

```
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:16436  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:0
          RX bytes:0 (0.0 B)  TX bytes:0 (0.0 B)

qg-c8c7efda-37 Link encap:Ethernet  HWaddr fa:16:3e:78:74:69
          inet addr:10.6.0.2  Bcast:10.7.255.255  Mask:255.254.0.0
          inet6 addr: fe80::f816:3eff:fe78:7469/64 Scope:Link
          UP BROADCAST RUNNING MULTICAST  MTU:1500  Metric:1
          RX packets:0 errors:0 dropped:0 overruns:0 frame:0
          TX packets:6 errors:0 dropped:0 overruns:0 carrier:0
          collisions:0 txqueuelen:0
          RX bytes:0 (0.0 B)  TX bytes:468 (468.0 B)

qr-8d90dc5a-8a Link encap:Ethernet  HWaddr fa:16:3e:61:59:75
          inet addr:172.16.1.1  Bcast:172.16.1.255  Mask:255.255.255.0
          inet6 addr: fe80::f816:3eff:fe61:5975/64 Scope:Link
          UP BROADCAST RUNNING MULTICAST  MTU:1500  Metric:1
          RX packets:173 errors:0 dropped:0 overruns:0 frame:0
          TX packets:6 errors:0 dropped:0 overruns:0 carrier:0
          collisions:0 txqueuelen:0
          RX bytes:55706 (55.7 KB)  TX bytes:468 (468.0 B)
```

As we can see the router was autmatically assigned the default gateway address on the `172.16.1.0/24` subnet i.e. `172.16.1.1`. It was also automatically assigned an IP address on the subnet associated with the "HPCS_private" network.

However, given the particular characteristics of our setup this failed to produce a working environment.

9. Limitations

One limitation in our setup was the limited functionality of the Layer 3 agent. Currently the L3 agent currently requires that "the node running quantum-l3-agent should not have an IP address manually configured on the NIC connected to the external network". However, in our installation the underlying networks themselves were themselves instantiated using other mechanisms -- namely SSH-based VPNs. As such the current approach of the Quantum L3 agent where these interfaces are automatically created and automatically assigned an IP address from respective subnets, with no administrative control of the process, proved to be too coarse for our environment.

As a result we were unable to provision connectivity to external resources as part of our setup. A side effect of this inability is that we were not able to use the new Nova metadata service providing access to VM instance metadata (as we noted in the **Node role and configuration** section) and, as mentioned, not being able to use other Operating Systems -- in particular "Ubuntu" versions of Linux -- that require access to this service during the bootup process. That is why in our setup we only limited to CirrOS and not test other versions of Linux that require this service.

While we do understand that such operations -- e.g. port creation / deletion, assiging IP and MAC addresses to those, etc. -- must be under Quantum control, more fine-grained mechanisms for controlling those constructs would be desirable. For example a mechanism that imports in Quantum network interfaces that are pre-existing in the underlying OS (together with their pre-defined characteristics like MAC and IP addresses, etc.) and allows them to be used "as such" in further building of network topologies would provide more fine-grained control on how Quantum models and constructs are mapped to the underlying infrastructure.

10. Acknowledgements

The author would like to thank Brent Salisbury for his invaluable comments and feedback. Without his encouragement and inspiration this work would have not been realized. I would also like to thank Endre Karlson for his gracious help in understanding the "less well documented" features and functions in Quantum. Last but not least I am indebted to the wonderful OpenStack and Quantum communities for their efforts and making this possible to begin with.

Appendix A -- Common prerequisites. Installing and configuring Keystone on Controller node

First, get some prerequisites in place (**Please note that this section is common for all the nodes in our installation and must be performed prior to installing any OpenStack services or tools on any of those nodes**):

```
# Make sure we're using the OpenStack services from Ubuntu Cloud Archive. Particularly important are dependent packages like e.g. "libvirt" and
```

```
#
echo "deb http://ubuntu-cloud.archive.canonical.com/ubuntu precise-updates/folsom main" >> /etc/apt/sources.list
apt-get update ; apt-get -y upgrade

# Install and configure NTP -- all the nodes in our installation need to be in sync
apt-get -y install ntp  ntpdate

cat <<NTP_CNF >/etc/ntp.conf
server ntp.ubuntu.com iburst
server 127.127.1.0
fudge 127.127.1.0 stratum 10

NTP_CNF

/etc/init.d/ntp stop
ntpdate ntp.ubuntu.com
/etc/init.d/ntp start
```

While the above steps are needed for all the nodes in our setup, there are some additional packagages and configuration specific to the Controller node. In particular, we need to install and configure RabbitMQ (used as a message queue by various services) and MySQL server itself (we will be use the same server instance on the Controller node for all the various databases in our setup):

```
# RabbitMQ: Install the server and reset the password for user "guest"
apt-get -y install rabbitmq-server
rabbitmqctl change_password guest guest
/etc/init.d/rabbitmq-server restart

# Configure and install MySQL server
MYSQLPASS=mysqlpasswd
cat <<MYSQL_PRESEED | debconf-set-selections
mysql-server-5.1 mysql-server/root_password password $MYSQLPASS
mysql-server-5.1 mysql-server/root_password_again password $MYSQLPASS
mysql-server-5.1 mysql-server/start_on_boot boolean true

MYSQL_PRESEED

# MySQL Get mysql-server and some packages that we will need
apt-get -y install curl python2.7 git make python-sphinx python-setuptools python-dev libxml2-dev libxslt1-dev libsasl2-dev libsqlite3-dev libss

#Reconfigure MySQL server to allow connections from other hosts as well
sed -i 's/127.0.0.1/0.0.0.0/g' /etc/mysql/my.cnf
service mysql restart
```

At this point we can create the MySQL database to be used for storing Keystone data, and a dedicated user to access that database:

```
mysql -u root -pmysqlpasswd -h 10.7.68.179 -e "create database keystone;"
mysql -u root -pmysqlpasswd -h 10.7.68.179 -e "create user 'keystonedbadmin'@'%' identified by 'keystonedbadminpasswd';"
mysql -u root -pmysqlpasswd -h 10.7.68.179 -e "grant all privileges on keystone.* to 'keystonedbadmin'@'%';"
```

Install Keystone and then (re)configure it (we will not be using the packaged configuration files):

```
apt-get -y install keystone python-keystone python-keystoneclient

# Stop and reconfigure keystone -- We will be using our own configuration files instead of the packaged-supplied versions
/etc/init.d/keystone stop

mv /etc/keystone /etc/keystone-ORIG
mkdir -p /etc/keystone

# First, "policy.json" for defining the admin role

cat <<POLICYJSON >/etc/keystone/policy.json
{
    "admin_required": [["role:admin"], ["is_admin:1"]]
}
POLICYJSON

# Then "keystone.conf"

cat <<KEYSTONECNF >/etc/keystone/keystone.conf
[DEFAULT]
```

```
bind_host = 0.0.0.0
public_port = 5000
admin_port = 5001
admin_token = 999888777666
compute_port = 8774
verbose = True
debug = True
#log_config = /etc/keystone/logging.conf

# ================= Syslog Options ============================
# Send logs to syslog (/dev/log) instead of to file specified
# by "log-file"
use_syslog = False
# Facility to use. If unset defaults to LOG_USER.
# syslog_log_facility = LOG_LOCAL0

[sql]
connection = mysql://keystonedbadmin:keystonedbadminpasswd@10.7.68.179/keystone
idle_timeout = 30
min_pool_size = 5
max_pool_size = 10
pool_timeout = 200

[identity]
driver = keystone.identity.backends.sql.Identity
[catalog]
driver = keystone.catalog.backends.sql.Catalog

[token]
driver = keystone.token.backends.sql.Token
# Amount of time a token should remain valid (in seconds). 86400 secs == 1440 mins == 24 hrs
expiration = 86400

[policy]
driver = keystone.policy.backends.rules.Policy

[ec2]
driver = keystone.contrib.ec2.backends.sql.Ec2

[filter:debug]
paste.filter_factory = keystone.common.wsgi:Debug.factory

[filter:token_auth]
paste.filter_factory = keystone.middleware:TokenAuthMiddleware.factory

[filter:admin_token_auth]
paste.filter_factory = keystone.middleware:AdminTokenAuthMiddleware.factory

[filter:json_body]
paste.filter_factory = keystone.middleware:JsonBodyMiddleware.factory

[filter:crud_extension]
paste.filter_factory = keystone.contrib.admin_crud:CrudExtension.factory

[filter:ec2_extension]
paste.filter_factory = keystone.contrib.ec2:Ec2Extension.factory

[app:public_service]
paste.app_factory = keystone.service:public_app_factory

[app:admin_service]
paste.app_factory = keystone.service:admin_app_factory

[pipeline:public_api]
pipeline = token_auth admin_token_auth json_body debug ec2_extension public_service

[pipeline:admin_api]
pipeline = token_auth admin_token_auth json_body debug ec2_extension crud_extension admin_service

[app:public_version_service]
paste.app_factory = keystone.service:public_version_app_factory

[app:admin_version_service]
paste.app_factory = keystone.service:admin_version_app_factory

[pipeline:public_version_api]
pipeline = public_version_service
```

```
[pipeline:admin_version_api]
pipeline = admin_version_service

[composite:main]
use = egg:Paste#urlmap
/v2.0 = public_api
/ = public_version_api

[composite:admin]
use = egg:Paste#urlmap
/v2.0 = admin_api
/ = admin_version_api

KEYSTONECNF
```

Now we can start Keystone and initialize its database:

```
# This will lock the terminal since in the configuration file we turned on debugging
/usr/bin/keystone-all init

#Initialize the DB
keystone-manage db_sync
```

Optionally, we can create a shell alias for admin-level Keystone commands. Please note that it is actually this shell alias instead of the "keystone" CLI command itself that we are using in the main text when we are creating tenants, roles, users, service endpoints, etc.

```
alias keystone='keystone --token 999888777666 --endpoint http://10.7.68.179:5001/v2.0/'
```

Appendix B -- Installing and configuring Swift on Controller node

First, we assume that all the installation and configuration steps detailed in **Appendix A** are in place on the Controller node.

For our purpose we will be using Swift object store located under the directory "/mnt/swift1" on the Controller server. Since in our case the Controller node is a VM itself with only ephemeral storage as local disk "/mnt/swift1" is where we have mounted an external block storage device to provide persistent storage device.

First, some preparations:

```
mkdir -p /srv /var/cache/swift/ /var/run/swift
cd /mnt/swift1
mkdir -p 1 2 3 4
cd /srv/
ln -s /mnt/swift1/1
ln -s /mnt/swift1/2
ln -s /mnt/swift1/3
ln -s /mnt/swift1/4

mkdir -p /etc/swift/object-server /etc/swift/container-server /etc/swift/account-server
mkdir -p /srv/1/node/sdb1 /srv/2/node/sdb2 /srv/3/node/sdb3 /srv/4/node/sdb4

useradd -d /tmp -c "Swift object storage" -U -s /bin/false swift
chown -L -R swift:swift /var/run/swift /var/cache/swift /etc/swift /srv
```

Before configuring Swift itself we need to configure "rsyncd" and (re)start it:

```
cat <<RSYNCEOF >/etc/rsyncd.conf
uid = swift
gid = swift
log file = /var/log/rsyncd.log
pid file = /var/run/rsyncd.pid
address = 127.0.0.1

[account6012]
max connections = 25
```

```
path = /srv/1/node/
read only = false
lock file = /var/lock/account6012.lock

[account6022]
max connections = 25
path = /srv/2/node/
read only = false
lock file = /var/lock/account6022.lock

[account6032]
max connections = 25
path = /srv/3/node/
read only = false
lock file = /var/lock/account6032.lock

[account6042]
max connections = 25
path = /srv/4/node/
read only = false
lock file = /var/lock/account6042.lock

[container6011]
max connections = 25
path = /srv/1/node/
read only = false
lock file = /var/lock/container6011.lock

[container6021]
max connections = 25
path = /srv/2/node/
read only = false
lock file = /var/lock/container6021.lock

[container6031]
max connections = 25
path = /srv/3/node/
read only = false
lock file = /var/lock/container6031.lock

[container6041]
max connections = 25
path = /srv/4/node/
read only = false
lock file = /var/lock/container6041.lock

[object6010]
max connections = 25
path = /srv/1/node/
read only = false
lock file = /var/lock/object6010.lock

[object6020]
max connections = 25
path = /srv/2/node/
read only = false
lock file = /var/lock/object6020.lock

[object6030]
max connections = 25
path = /srv/3/node/
read only = false
lock file = /var/lock/object6030.lock

[object6040]
max connections = 25
path = /srv/4/node/
read only = false
lock file = /var/lock/object6040.lock

RSYNCEOF

# Enable and restart rsyncd
sed -i 's/RSYNC_ENABLE=false/RSYNC_ENABLE=true/g' /etc/default/rsync
service rsync restart
```

For Swift configuration proper, first we configure a Swift hash:

```
cat <<SWIFTEOF >/etc/swift/swift.conf
[swift-hash]
# random unique (preferably alphanumeric) string that can never change (DO NOT LOSE).

swift_hash_path_suffix = F4761F094B

SWIFTEOF
```

Then configure the Swift account server (via "/etc/swift/account-server.conf" and "/etc/swift/account-server/{1,2,3,4}.conf"):

```
cat <<EOF >/etc/swift/account-server.conf
[DEFAULT]
bind_ip = 0.0.0.0
workers = 2

[pipeline:main]
pipeline = account-server

[app:account-server]
use = egg:swift#account

[account-replicator]

[account-auditor]

[account-reaper]

EOF

#
mkdir /etc/swift/account-server/
#

cat <<EOF >/etc/swift/account-server/1.conf
[DEFAULT]
devices = /srv/1/node
mount_check = false
bind_port = 6012
user = swift
log_facility = LOG_LOCAL2

[pipeline:main]
pipeline = account-server

[app:account-server]
use = egg:swift#account

[account-replicator]
vm_test_mode = yes

[account-auditor]

[account-reaper]

EOF
#
cd /etc/swift/account-server
#
sed -e 's/srv\/1/srv\/2/g' -e 's/6012/6022/g' -e 's/LOG_LOCAL2/LOG_LOCAL3/g' <1.conf >2.conf
sed -e 's/srv\/1/srv\/3/g' -e 's/6012/6032/g' -e 's/LOG_LOCAL2/LOG_LOCAL4/g' <1.conf >3.conf
sed -e 's/srv\/1/srv\/4/g' -e 's/6012/6042/g' -e 's/LOG_LOCAL2/LOG_LOCAL5/g' <1.conf >4.conf
```

The container server is configured using "/etc/swift/container-server.conf" and "/etc/swift/container-server/{1,2,3,4}.conf":

```
cat <<EOF >/etc/swift/container-server.conf
[DEFAULT]
bind_ip = 0.0.0.0
workers = 2

[pipeline:main]
pipeline = container-server

[app:container-server]
use = egg:swift#container

[container-replicator]
```

```
[container-updater]

[container-auditor]

[container-sync]

EOF

mkdir /etc/swift/container-server/

cat <<EOF >/etc/swift/container-server/1.conf
[DEFAULT]
devices = /srv/1/node
mount_check = false
bind_port = 6011
user = swift
log_facility = LOG_LOCAL2

[pipeline:main]
pipeline = container-server

[app:container-server]
use = egg:swift#container

[container-replicator]
vm_test_mode = yes

[container-updater]

[container-auditor]

[container-sync]

EOF

cd /etc/swift/container-server

sed -e 's/srv\/1/srv\/2/g' -e 's/6011/6021/g' -e 's/LOG_LOCAL2/LOG_LOCAL3/g' <1.conf >2.conf
sed -e 's/srv\/1/srv\/3/g' -e 's/6011/6031/g' -e 's/LOG_LOCAL2/LOG_LOCAL4/g' <1.conf >3.conf
sed -e 's/srv\/1/srv\/4/g' -e 's/6011/6041/g' -e 's/LOG_LOCAL2/LOG_LOCAL5/g' <1.conf >4.conf
```

The Swift object server configuration is defined in "/etc/swift/object-server.conf" and "/etc/swift/object-server/{1,2,3,4}.conf”:

```
cat <<EOF >/etc/swift/object-server.conf
[DEFAULT]
bind_ip = 0.0.0.0
workers = 2

[pipeline:main]
pipeline = object-server

[app:object-server]
use = egg:swift#object

[object-replicator]

[object-updater]

[object-auditor]

EOF

mkdir -p /etc/swift/object-server

cat <<EOF >/etc/swift/object-server/1.conf
[DEFAULT]
devices = /srv/1/node
mount_check = false
bind_port = 6010
user = swift
log_facility = LOG_LOCAL2

[pipeline:main]
pipeline = object-server
```

```
[app:object-server]
use = egg:swift#object

[object-replicator]
vm_test_mode = yes

[object-updater]

[object-auditor]

EOF

cd /etc/swift/object-server/

sed -e 's/srv\/1/srv\/2/g' -e 's/6010/6020/g' -e 's/LOG_LOCAL2/LOG_LOCAL3/g' <1.conf >2.conf
sed -e 's/srv\/1/srv\/3/g' -e 's/6010/6030/g' -e 's/LOG_LOCAL2/LOG_LOCAL4/g' <1.conf >3.conf
sed -e 's/srv\/1/srv\/4/g' -e 's/6010/6040/g' -e 's/LOG_LOCAL2/LOG_LOCAL5/g' <1.conf >4.conf
```

Build, verify and re-balance the Swift rings:

```
# Build the rings
cd /etc/swift

swift-ring-builder object.builder create 18 3 1
swift-ring-builder object.builder add z1-127.0.0.1:6010/sdb1 1
swift-ring-builder object.builder add z2-127.0.0.1:6020/sdb2 1
swift-ring-builder object.builder add z3-127.0.0.1:6030/sdb3 1
swift-ring-builder object.builder add z4-127.0.0.1:6040/sdb4 1


swift-ring-builder container.builder create 18 3 1
swift-ring-builder container.builder add z1-127.0.0.1:6011/sdb1 1
swift-ring-builder container.builder add z2-127.0.0.1:6021/sdb2 1
swift-ring-builder container.builder add z3-127.0.0.1:6031/sdb3 1
swift-ring-builder container.builder add z4-127.0.0.1:6041/sdb4 1


swift-ring-builder account.builder create 18 3 1
swift-ring-builder account.builder add z1-127.0.0.1:6012/sdb1 1
swift-ring-builder account.builder add z2-127.0.0.1:6022/sdb2 1
swift-ring-builder account.builder add z3-127.0.0.1:6032/sdb3 1
swift-ring-builder account.builder add z4-127.0.0.1:6042/sdb4 1

# Verify the rings
swift-ring-builder account.builder
swift-ring-builder container.builder
swift-ring-builder object.builder

# Re-balance the rings (this may take a while)
swift-ring-builder account.builder rebalance
swift-ring-builder container.builder rebalance
swift-ring-builder object.builder rebalance
```

Final piece of Swift configuration is the Swift proxy:

```
cat <<SWIFTPROXYEOF >/etc/swift/proxy-server.conf
[DEFAULT]
bind_port = 8080
user = swift

[pipeline:main]
pipeline = catch_errors healthcheck cache authtoken swiftauth proxy-server

[app:proxy-server]
use = egg:swift#proxy
allow_account_management = true
account_autocreate = true
set log_name = swift-proxy-server
set log_facility = LOG_LOCAL0
set log_level = DEBUG
set access_log_name = swift-proxy-server
set access_log_facility = LOG_LOCAL0
set access_log_level = DEBUG
set log_headers = True

[filter:healthcheck]
use = egg:swift#healthcheck

[filter:catch_errors]
use = egg:swift#catch_errors

[filter:cache]
use = egg:swift#memcache
set log_name = cache

[filter:swiftauth]
paste.filter_factory = keystone.middleware.swift_auth:filter_factory
operator_roles = admin, SwiftOperator
is_admin = true

[filter:authtoken]
paste.filter_factory = keystone.middleware.auth_token:filter_factory
service_protocol = http
service_host = 10.7.68.179
service_port = 5000
auth_protocol = http
auth_host = 10.7.68.179
auth_port = 5001
admin_tenant_name = service
admin_user = swiftadmin
admin_password = swiftadminpasswd
delay_auth_decision = 0
signing_dir = /etc/swift

SWIFTPROXYEOF
```

At this point we can start the Swift services themselves:

```
chown -R swift:swift /etc/swift

swift-init main start
swift-init rest start
```

The last piece of configuration is putting in place a shell alias that we will be using instead of the Swift CLI command itself:

```
alias swift='swift -v -V 2.0 -A http://10.7.68.179:5000/v2.0/ -U service:swiftadmin -K swiftadminpasswd'
```

Appendix C -- Installing and configuring Glance with Swift back-end on Controller node

As we will be using Swift as storage back-end for Glance we assume that Swift is installed and configured on the Controller server according to the instructions in **Appendix B**. Wiith that in place we can proceed to install Glance from the packages in the Ubuntu Cloud Archive:

```
apt-get -y install glance glance-api python-glanceclient glance-common
```

We will be storing the Glance registry data in dedicated MySQL database, which will be located on the same MySQL server instance running on the Controller node. For that purpose we will create that database and a dedicate user to access it:

```
mysql -u root -pmysqlpasswd -h 10.7.68.179 -e "create database glance;"
mysql -u root -pmysqlpasswd -h 10.7.68.179 -e "create user 'glancedbadmin'@'%' identified by 'glancedbadminpasswd';"
mysql -u root -pmysqlpasswd -h 10.7.68.179 -e "grant all privileges on glance.* to 'glancedbadmin'@'%';"
```

First Glance component that we will be configuring is the Glance API server (via "/etc/glance/glance-api.conf" and "/etc/glance/glance-api-paste.ini"):

```
######## Glance API configuration file "/etc/glance/glance-api.conf"

cat <<GLANCEAPICONF >/etc/glance/glance-api.conf
[DEFAULT]
# Show more verbose log output (sets INFO log level output)
```

```
verbose = True

# Show debugging output in logs (sets DEBUG log level output)
debug = True

# Which backend store should Glance use by default is not specified
# in a request to add a new image to Glance? Default: 'file'
# Available choices are 'file', 'swift', and 's3'
default_store = swift

# Default admin role is "glanceadmin"
admin_role = glanceadmin

# Address to bind the API server
bind_host = 0.0.0.0

# Port the bind the API server to
bind_port = 9292

# Log to this file. Make sure you do not set the same log
# file for both the API and registry servers!
# log_file = /var/log/glance/api.log

# Backlog requests when creating socket
backlog = 4096

# Number of Glance API worker processes to start.
# On machines with more than one CPU increasing this value
# may improve performance (especially if using SSL with
# compression turned on). It is typically recommended to set
# this value to the number of CPUs present on your machine.
workers = 2

# ================= Syslog Options ============================

# Send logs to syslog (/dev/log) instead of to file specified
# by 'log_file'
use_syslog = False

# Facility to use. If unset defaults to LOG_USER.
# syslog_log_facility = LOG_LOCAL0

# ============ Registry Options ===============================

# Address to find the registry server
registry_host = 0.0.0.0

# Port the registry server is listening on
registry_port = 9191

# What protocol to use when connecting to the registry server?
# Set to https for secure HTTP communication
registry_client_protocol = http

# ============ Notification System Options =====================

# Notifications can be sent when images are create, updated or deleted.
# There are three methods of sending notifications, logging (via the
# log_file directive), rabbit (via a rabbitmq queue), qpid (via a Qpid
# message queue), or noop (no notifications sent, the default)
notifier_strategy = rabbit

# Configuration options if sending notifications via rabbitmq (these are
# the defaults)
rabbit_host = 10.7.68.179
rabbit_port = 5672
rabbit_use_ssl = false
rabbit_userid = guest
rabbit_password = guest
rabbit_virtual_host = /
rabbit_notification_exchange = glance
rabbit_notification_topic = glance_notifications

# ============ Swift Store Options =============================

# Address where the Swift authentication service lives
# Valid schemes are 'http://' and 'https://'
# If no scheme specified,  default to 'https://'
#
# OBS: When using Keystone it is actually the Keystone service port (normally tcp/5000)
#
swift_store_auth_address = http://10.7.68.179:5000/v2.0/

# Ensure we'll be using version 2 when authenticating with Keystone
swift_store_auth_version = 2.0

# User to authenticate against the Swift authentication service
# If you use Swift authentication service, set it to 'account':'user'
# where 'account' is a Swift storage account and 'user'
# is a user in that account
swift_store_user = service:glanceadmin

# Auth key (i.e. password, "swift -K" option) for the user authenticating against the
# Swift authentication service
swift_store_key = glanceadminpasswd

# Container within the account that the account should use
# for storing images in Swift
swift_store_container = Glance:

# Do we create the container if it does not exist?
swift_store_create_container_on_put = True

# What size, in MB, should Glance start chunking image files
# and do a large object manifest in Swift? By default, this is
# the maximum object size in Swift, which is 5GB
swift_store_large_object_size = 5120

# When doing a large object manifest, what size, in MB, should
# Glance write chunks to Swift? This amount of data is written
# to a temporary disk buffer during the process of chunking
# the image file, and the default is 200MB
swift_store_large_object_chunk_size = 200


# ============ Delayed Delete Options =============================

# Turn on/off delayed delete
delayed_delete = False

# Delayed delete time in seconds
scrub_time = 43200

# Directory that the scrubber will use to remind itself of what to delete
# Make sure this is also set in glance-scrubber.conf
scrubber_datadir = /var/lib/glance/scrubber

# =============== Image Cache Options =============================

### (XXX): Note that some  options need to be both here and in "glance-cache.conf" / "glance-cache-paste.ini" -- and consistent...

# Base directory that the Image Cache uses
# OBS:  As we will be using the "xattr" driver instead of the default "sqlite" make sure this is on top of a filesystem that supports "xattr" (e
image_cache_driver = xattr
image_cache_dir = /mnt/swift1/glance-image-cache/

# Using keystone with caching ===> The relevant application pipeline in  "glance-api-paste.ini" is  "[pipeline:glance-api-keystone+caching]"
[paste_deploy]
flavor = keystone+caching

GLANCEAPICONF

######## Glance API Configuration file "/etc/glance/glance-api-paste.ini"

cat <<GLANCEAPIPASTE >/etc/glance/glance-api-paste.ini
# Use the following pipeline for keystone auth with caching
# Enabled by adding in glance-api.conf:
#   [paste_deploy]
#   flavor = keystone+caching
#
[pipeline:glance-api-keystone+caching]
pipeline = versionnegotiation authtoken context cache rootapp

[filter:versionnegotiation]
```

```
paste.filter_factory = glance.api.middleware.version_negotiation:VersionNegotiationFilter.factory

[filter:authtoken]
paste.filter_factory = keystone.middleware.auth_token:filter_factory
service_protocol = http
service_host = 10.7.68.179
service_port = 5000
auth_protocol = http
auth_host = 10.7.68.179
auth_port = 5001
admin_tenant_name = service
admin_user = glanceadmin
admin_password = glanceadminpasswd
delay_auth_decision = 0
signing_dir = /etc/glance

[filter:context]
paste.filter_factory = glance.api.middleware.context:ContextMiddleware.factory

[filter:cache]
paste.filter_factory = glance.api.middleware.cache:CacheFilter.factory

[composite:rootapp]
paste.composite_factory = glance.api:root_app_factory
/: apiversions
/v1: apiv1app
/v2: apiv2app

[app:apiversions]
paste.app_factory = glance.api.versions:create_resource

[app:apiv1app]
paste.app_factory = glance.api.v1.router:API.factory

[app:apiv2app]
paste.app_factory = glance.api.v2.router:API.factory

GLANCEAPIPASTE
```

Next we will configure the Glance cache ("/etc/glance/glance-cache.conf" and "glance-cache-paste.ini"). Please note that the Glance image cache is located under "/mnt/swift1/glance-image-cache/" i.e. it is located on the same block storage device that we used for our Swift object store (but it is distinct from the latter):

```
######## Glance Cache configuration file "/etc/glance/glance-cache.conf"

cat <<GLANCECACHE >/etc/glance/glance-cache.conf
[DEFAULT]
# Show more verbose log output (sets INFO log level output)
verbose = True

# Show debugging output in logs (sets DEBUG log level output)
debug = False

log_file = /var/log/glance/image-cache.log

# Send logs to syslog (/dev/log) instead of to file specified by `log_file`
use_syslog = False

# Directory that the Image Cache writes data to
image_cache_dir = /var/lib/glance/image-cache/

# Number of seconds after which we should consider an incomplete image to be
# stalled and eligible for reaping
image_cache_stall_time = 86400

# image_cache_invalid_entry_grace_period - seconds
#
# If an exception is raised as we're writing to the cache, the cache-entry is
# deemed invalid and moved to <image_cache_datadir>/invalid so that it can be
# inspected for debugging purposes.
#
# This is number of seconds to leave these invalid images around before they
# are elibible to be reaped.
image_cache_invalid_entry_grace_period = 3600

# Max cache size in bytes
```

```
image_cache_max_size = 10737418240

# Address to find the registry server
registry_host = 0.0.0.0

# Port the registry server is listening on
registry_port = 9191

# Auth settings if using Keystone
auth_url = http://10.7.68.179:5001/v2.0/
admin_tenant_name = service
admin_user = glanceadmin
admin_password = glanceadminpasswd
delay_auth_decision = 0
signing_dir = /etc/glance

GLANCECACHE

######## Glance Cache configuration file "/etc/glance/glance-cache-paste.ini"

cat <<GLANCECACHEPASTE >/etc/glance/glance-cache-paste.ini
[app:glance-pruner]
paste.app_factory = glance.common.wsgi:app_factory
glance.app_factory = glance.image_cache.pruner:Pruner

[app:glance-prefetcher]
paste.app_factory = glance.common.wsgi:app_factory
glance.app_factory = glance.image_cache.prefetcher:Prefetcher

[app:glance-cleaner]
paste.app_factory = glance.common.wsgi:app_factory
glance.app_factory = glance.image_cache.cleaner:Cleaner

[app:glance-queue-image]
paste.app_factory = glance.common.wsgi:app_factory
glance.app_factory = glance.image_cache.queue_image:Queuer

GLANCECACHEPASTE
```

The Glance image scrubber application is configured via "/etc/glance/glance-scrubber.conf" and respectively "/etc/glance/glance-scrubber-paste.ini":

```
######## Glance Scrubber configuration file "/etc/glance/glance-scrubber.conf"

cat <<GLANCESCRUBBER >/etc/glance/glance-scrubber.conf
[DEFAULT]
# Show more verbose log output (sets INFO log level output)
verbose = True

# Show debugging output in logs (sets DEBUG log level output)
debug = False

# Log to this file. Make sure you do not set the same log
# file for both the API and registry servers!
log_file = /var/log/glance/scrubber.log

# Send logs to syslog (/dev/log) instead of to file specified by `log_file`
use_syslog = False

# Should we run our own loop or rely on cron/scheduler to run us
daemon = False

# Loop time between checking for new items to schedule for delete
wakeup_time = 300

# Directory that the scrubber will use to remind itself of what to delete
# Make sure this is also set in glance-api.conf
scrubber_datadir = /var/lib/glance/scrubber

# Only one server in your deployment should be designated the cleanup host
cleanup_scrubber = False

# pending_delete items older than this time are candidates for cleanup
cleanup_scrubber_time = 86400

# Address to find the registry server for cleanups
```

```
registry_host = 0.0.0.0

# Port the registry server is listening on
registry_port = 9191

GLANCESCRUBBER

######## Glance Scrubber configuration file "/etc/glance/glance-scrubber-paste.ini"

cat <<GLANCESCRUBBERPASTE >/etc/glance/glance-scrubber-paste.ini
[app:glance-scrubber]
paste.app_factory = glance.common.wsgi:app_factory
glance.app_factory = glance.store.scrubber:Scrubber

GLANCESCRUBBERPASTE
```

The Glance registry configuration files are "/etc/glance/glance-registry.conf" and respectively "/etc/glance/glance-registry-paste.ini"

```
######## Glance Registry configuration file "/etc/glance/glance-registry.conf"

cat <<GLANCEREGISTRY >/etc/glance/glance-registry.conf
[DEFAULT]
# Show more verbose log output (sets INFO log level output)
verbose = True

# Show debugging output in logs (sets DEBUG log level output)
debug = True

# Address to bind the registry server
bind_host = 0.0.0.0

# Port the bind the registry server to
bind_port = 9191

# Log to this file. Make sure you do not set the same log
# file for both the API and registry servers!
log_file = /var/log/glance/registry.log

# Backlog requests when creating socket
backlog = 4096

# SQLAlchemy connection string for the reference implementation
# registry server. Any valid SQLAlchemy connection string is fine.
# See: http://www.sqlalchemy.org/docs/05/reference/sqlalchemy/connections.html#sqlalchemy.create_engine
sql_connection = mysql://glancedbadmin:glancedbadminpasswd@10.7.68.179/glance

# Period in seconds after which SQLAlchemy should reestablish its connection
# to the database.
#
# MySQL uses a default `wait_timeout` of 8 hours, after which it will drop
# idle connections. This can result in 'MySQL Gone Away' exceptions. If you
# notice this, you can lower this value to ensure that SQLAlchemy reconnects
# before MySQL can drop the connection.
sql_idle_timeout = 3600

# Limit the api to return `param_limit_max` items in a call to a container. If
# a larger `limit` query param is provided, it will be reduced to this value.
api_limit_max = 1000

# If a `limit` query param is not provided in an api request, it will
# default to `limit_param_default`
limit_param_default = 25

# ================= Syslog Options ============================

# Send logs to syslog (/dev/log) instead of to file specified
# by `log_file`
use_syslog = False

# Using keystone  ===> The relevant application pipeline in "glance-registry-paste.ini" is  "[pipeline:glance-registry-keystone]"
[paste_deploy]
flavor = keystone

GLANCEREGISTRY
```

```
######## Glance Registry configuration file "/etc/glance/glance-registry-paste.ini"

cat <<GLANCEREGISTRYPASTE >/etc/glance/glance-registry-paste.ini

# Use the following pipeline for keystone auth
# Enabled by adding in glance-registry.conf:
#   [paste_deploy]
#   flavor = keystone
#
[pipeline:glance-registry-keystone]
pipeline = authtoken context registryapp

[filter:authtoken]
paste.filter_factory = keystone.middleware.auth_token:filter_factory
service_protocol = http
service_host = 10.7.68.179
service_port = 5000
auth_protocol = http
auth_host = 10.7.68.179
auth_port = 5001
admin_tenant_name = service
admin_user = glanceadmin
admin_password = glanceadminpasswd
delay_auth_decision = 0
signing_dir = /etc/glance

[filter:context]
paste.filter_factory = glance.api.middleware.context:ContextMiddleware.factory

[app:registryapp]
paste.app_factory = glance.registry.api.v1:API.factory

GLANCEREGISTRYPASTE
```

A final Glance configuration file is "/etc/glance/policy.json":

```
cat <<GLANCEPOLICYJSON >/etc/glance/policy.json
{
    "default": [],
    "manage_image_cache": [["role:admin"]]
}

GLANCEPOLICYJSON
```

We can now start the Glance image services:

```
# First, some preparations
mkdir -p  /var/log/glance /mnt/swift1/glance-image-cache/

# The first Glance service to be started is Glance API. Please note that we are starting glance API with verbose debugging turned on, so this wi
glance-api /etc/glance/glance-api.conf --debug |& tee /var/log/glance/glance-api.log

# The second Glance service to start is the Glance registry. Just like Glance API above, this will lock the terminal
glance-registry /etc/glance/glance-registry.conf --debug |& tee /var/log/glance/glance-registry.log
```

One last configuration item is establishing a shell alias for administrative access to Glance image services:

```
alias glance='glance -I glanceadmin -K glanceadminpasswd -T service -S keystone -N http://10.7.68.179:5000/v2.0'
```

One final note: When we are using the "glance" command in this document we are referring to the shell alias above rather then the Glance CLI command itself.