Multiple L3 Backends in a cloud

Manjeet Singh Bhatia<manjeet.s.bhatia@intel.com>(Intel),
Isaku yamahata<isaku.yamahata@intel.com>(Intel)
Takashi Yamamoto<yamamoto@midokura.com>(Midokura)

OpenStack Summit Vancouver: May 22, 2018
Agenda

- Neutron L3 flavors framework
- Why L3 flavors?
- Use case
- Driver Enabling
- Sample L3 driver for a backend.
- Traffic b/w different backends
- Datapath connectivity among backends (pie in the sky)
- Challenges
- Summary
Neutron L3 flavor framework

L3 Flavor Framework:

- Single L3 Router Plugin with flavor support
  - with Neutron callbacks
  - Instead of backend specific L3 plugin
- Allows multiple L3 backends
  - User specifies flavor which L3 backends to use
  - Backends implements L3 flavor driver
- Router instance is associated with flavor=l3 backend
Neutron L3 Flavors framework!

- Source: Neutron L3 Flavors Framework spec
- Neutron Flavors enables multiple L3 backends.
- Driver X can be used for subset of routers and Driver Y for another set of routers.
- It's similar to ML2 but there's an important difference.
Use cases

- Multiple backends in a single Neutron deployment, each has its own logical network topology, completely separated each other
- It would allow incremental migrations from one backend to the other
Another motivation: Simplification

- DB transaction Issue
  - L2 plugin has its own db transactions
  - L2 plugin, e.g. create_port, shouldn’t be called within a db transaction of L3 plugin

- Implementation consistency and code reduction
  - The reference L3 plugin has been refactored to avoid the above mentioned transaction issues.
  - It’s better for vendors to use the same framework instead of keeping to improve their own monolithic L3 plugins.

We all love clean code!
How to use vendor L3 flavor

- Use `router` as L3 service_plugin
- Specify your flavor as a **L3_ROUTER_NAT** service provider

```
service_plugins = router, xyz, ...

[service_providers]

  service_provider =
  L3_ROUTER_NAT:ODL:networking_odl.l3.l3_flavor.ODLL3ServiceProvider:default
```
How to use L3 flavor  (Cont.)

Prepare a flavor and its profile

1. `openstack network flavor profile create --driver networking_odl.l3.l3_flavor.ODLL3ServiceProvider`
2. `openstack network flavor create --service-type L3_ROUTER_NAT odl`
3. `openstack network flavor add profile odl <flavorprofileid>`

Create a router with the flavor

4. `neutron router-create router1 --flavor odl`
Sample L3 flavor driver
Sample L3 flavor Driver

```python
@registry.receives(resources.ROUTER, [events.PRECOMMIT_DELETE],
                     priority_group.PRIORITY_ROUTER_DRIVER)
@log_helpers.log_method
def router_delete_precommit(self, resource, event, trigger, **kwargs):
    """implementation -----------""

@registry.receives(resources.ROUTER, [events.AFTER_DELETE],
                     priority_group.PRIORITY_ROUTER_DRIVER)
@log_helpers.log_method
def router_delete_postcommit(self, resource, event, trigger, **kwargs):
    """implementation -----------""

@registry.receives(resources.FLOATING_IP, [events.PRECOMMIT_CREATE])
@log_helpers.log_method
@journal.call_thread_on_end
def floatingip_create_precommit(self, resource, event, trigger, **kwargs):
    """implementation -----------""

@registry.receives(resources.FLOATING_IP, [events.PRECOMMIT_UPDATE])
@log_helpers.log_method
@journal.call_thread_on_end
def floatingip_update_precommit(self, resource, event, trigger, **kwargs):
    """implementation -----------""

@registry.receives(resources.FLOATING_IP, [events.PRECOMMIT_DELETE_ASSOCIATIONS])
@log_helpers.log_method
@journal.call_thread_on_end
def floatingip_delete_precommit(self, resource, event, trigger, **kwargs):
    """implementation -----------""
```
Traffic between multiple backends: Pie in the sky

- East-west traffic between multiple L3 backends
- API wise, shared router connected to each L3 network or L2GW?
- Implementation wise: requires common router or gateway

Opens:
- Any requirements?
- Volunteers?

How?

- ODL Router
- ODL Network
- MidoNet Network
Traffic between multiple backends

- The simplest solution: Disallow such configurations
  - You can still provide connectivities using the other mechanisms.
- Use legacy L3-agent compatible port
  - Hopefully many of backends can support it trivially
- Design something distributed
  - Pie in the sky
  - More work for dubious usefulness
  - It’s actually more complicated
  - Floating-IP, A network can be backed by multiple backends, Live migration between backends (multiple port binding), Hierarchical port binding
Challenges

- There were missing notifications in neutron (needed a fix)
- Callback execution order was not guaranteed.
- Changes to neutron and neutron-lib.
Future work

- **FloatingIP compatibility**
  - Compatibility between L3 flavor and ML2 mech driver

- **More tests. Tempest**

- **Tenants associated to l3 flavor**
  - New tenants/user to use new backends
  - Existing tenants to use the existing backend for migration
Summary

- L3 flavor works and L3 flavor drivers are coming

Call for action

- test/use it
- Convert your L3 plugin into L3 flavor driver
Reference Code

1. https://review.openstack.org/#/c/523257/ (Adding callbacks to neutron)
2. https://review.openstack.org/#/c/504182/ (ODL L3 Flavor Driver)
3. https://review.openstack.org/#/c/544116/ (Functional tests)