Neutron Services' Insertion & Chaining Model, and API

Table of Contents

Document History

Version	Date	Authors	Changes
v0.1	Feb 25 2013	Sumit Naiksatam, Kanzhe Jiang	Initial Version
v0.11	March 22, 2013	Sumit Naiksatam	More specifics on service-types, presence (strict or loose) semantics
v0.2	Oct 6 2013	Sumit Naiksatam	Aligning with changes in Havana

	Oct 11 2013	Sumit Naiksatam	Incorporating discussion from F2F with Nachi, and Cisco and Dell
	Dec 13 2013	Sumit	Incorporating feedback from Icehouse summit discussion, and followup discussion with Nachi

Collaborators	
Sumit, Kanzhe, KC Wang, Kevin Benton - Big Switch Networks	Nachi Ueno - NTT
Sridar Kandaswamy, Joseph Swaminathan, David Chang - Cisco Systems	Eugene Nikanorov -Mirantis
Greg Regnier, Isaku Yamahata, Christian Maciocco, Intel	

1. Introduction

Neutron, as of the Havana release, supports three advanced services - Firewall (FWaaS), Loadbalancer (LBaaS), and VPN (VPNaaS) in addition to the core L2 and L3 functionality (the later has also been factored into a pluggable advanced service implementation). Each of these advanced services is either inserted implicitly into the tenant topology or explicitly associated with a neutron router. However, there is no generic service insertion mechanism to facilitate L3, L2, bump-in-the-wire, and tap insertion modes. Moreover, with more than one service, it becomes relevant to explore the model of how multiple services can be sequenced in a chain.

An example, in the context of today's reference implementations, is the insertion and chaining of firewall and VPN services. Each of these reference implementations rely on the use of IPTables chains to program the relevant filters and policies to achieve their respective functions. However, in the absence of a chaining abstraction to express the sequence of these services, these implementations act independently and the resulting order of operations is incidental and cannot be controlled.

In this proposal we will build on the existing notions of services and service_types (or more specifically the `service_provider` resource which is part of the `service_type` extension) to realize a tenant-facing service chaining and insertion model, and a corresponding contract to be implemented by the provider. One of the objectives of this proposal is to support both modes of instantiating services - as a single independent service, or as a part of a chain - with support for the former in a non-disruptive fashion (since this is the default mode today).

When a service chain is desired, the backend implementation needs to interpret this service chaining model and insert each service in the chain in the context of the service chain. This requires deep knowledge of the network topology (both, physical/provider and virtual/tenant) and is best driven through the Neutron network (L2/L3) plugin. In most cases the implementation might need to implement some level of traffic steering to be able to send the traffic sequentially through the service chain. This enables external orchestration entities, such as Heat, to explicitly define the interaction between networks and services.

2. Theory Of Operation

A network orchestrating entity is required in the system that understands the physical and virtual topologies and is subsequently able to correctly insert and chain the services. Today Neutron employs a combination of *core* and *service* plugins to realize the services. In this document we will refer to them collectively as a *plugin*.

Service Instances

In the proposed model it is assumed that the service vendor knows how to implement the service and provide logically separate ***service instances*** on request. For example, a user can create a `firewall` service instance using the FWaaS API. An instance of the service can manifest in one of the following ways:

1. The provider deploys a physical device, and the service runs on this device. E.g. A Loadbalancer or a Firewall appliance. In this case the physical connections are already present. The provider needs to provide a Neutron logical port to the tenant on which to request connecting the service instance.
2. The service uses a virtual appliance (VM); one per instance of the service, or share the VM between multiple service instances. In either case, the service requests creating a Neutron logical port(s) to attach the service.
3. The tenant spawns a VM that has the service implementation. There needs to be a corresponding service implementation that will allow the tenant to point to this VM. The service implementation will then take over as in case 2 above.
4. A host OS configuration (e.g. namespaces and IPTables).
5. An independent OS process (e.g. HAProxy).

The actual instantiation of the service instance is done by the corresponding service plugin. This in turn can be can be managed by a resource manager. See https://blueprints.launchpad.net/neutron/+spec/dynamic-network-resource-mgmt and https://blueprints.launchpad.net/neutron/+spec/quantum-service-scheduler for proposals related to resource manangement strategies and https://blueprints.launchpad.net/neutron/+spec/adv-services-in-vms for the VM instantiation mechanism.

Service Insertion Context

The service implementation also involves associating the service instance with a Neutron construct (such as subnet, or router) and serves as the ***service insertion context***. For example, for services associated with a router, the insertion context will point to the router's ID.

The service insertion context can be either expressed by the user (when a service is being instantiated independently) or constructed by the plugin based on the service chain's context. Thus based on the service insertion context we can support both modes -

- independent service insertion - wherein the service is instantiated and associated with Neutron construct(s) in one step and becomes immediately operational in the data plane on successful instantiation. This is the current mode of service operation and will continue to serve as the default mode.
- insertion as a part of a service chain - wherein the service is instantiated as a first step but is not operational in the data plane until it's made a part of a service chain. Once a chain is created with a reference to that service, the service is inserted based on the context of the chain and becomes operational on successful instantiation of the service chain.

Service Type

Neutron currently supports the notion of a ***service type*** as a part of the `service_provider` resource in the `service_type` extension (https://wiki.openstack.org/wiki/Neutron/ServiceTypeFramework). We will continue to leverage the `service_type` attribute (within the `service_provider` resource) to differentiate services. The following service types are possible:

1. Firewall (existing definition)
2. Loadbalancer (existing definition)
3. VPN (existing definition)

Others will be added as new services are defined.

The `service_provider` allows the binding of a particular ***service instance*** to a particular provider (vendor) implementation. The provider implementation makes use of the ***service insertion context*** to configure the data path for this particular service instance.

Service Chain

A service chain is a directed graph of service instance references. In it's simplest form it's an ordered sequence of service instance references. Each plugin/provider may decide which service chains to support by publishing them in a `service_chain_provider` configuration. The user would provide an insertion context for the entire chain. The insertion context for a particular service is then constructed by the provider based on the insertion context of the service chain.

The diagram below describes the association between the various resource entities involved in this model.

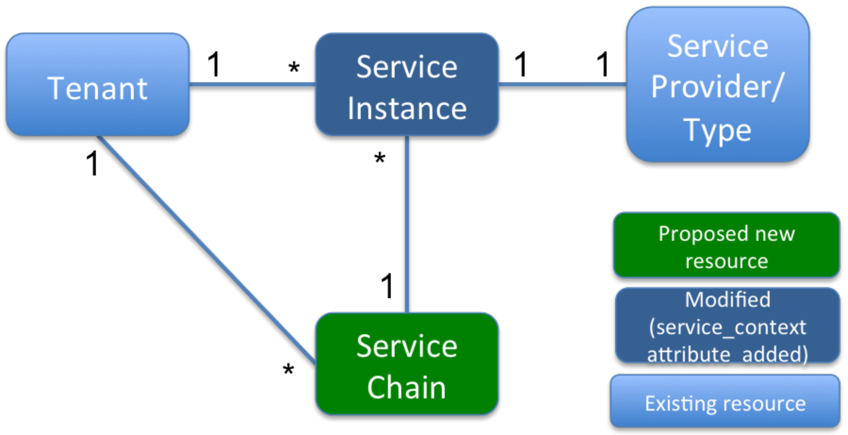


Figure 1 Entity Relationships

3. New/Modified Resources

1. ~~Service Insertion Context (service_insertion_context)~~

~~Attribute Name~~	~~Type~~	~~Default Value~~	~~Description~~
~~**id**~~	~~uuid-str~~	~~generated~~	~~UUID for the service_type~~
~~**tenant_id**~~	~~uuid-str~~	~~N/A~~	~~Owner of the service_type.~~
~~**name**~~	~~String~~	~~None~~	
~~**networks**~~	~~List of Uuids~~	~~None~~	
~~**subnets**~~	~~List of Uuids~~	~~None~~	
~~**routers**~~	~~List of uuids~~	~~None~~	
~~**ports**~~	~~List off uuids~~	~~None~~	
~~**tap**~~	~~boolean~~	~~False~~	
~~**in_chain**~~	~~boolean~~	~~False~~	
~~**description**~~	~~String~~	~~None~~	

1. Service Context Attribute in each Service instance

Instead of having the insertion_context as a separate resource, each service will have a service_context attribute, which will the following key-value pairs:

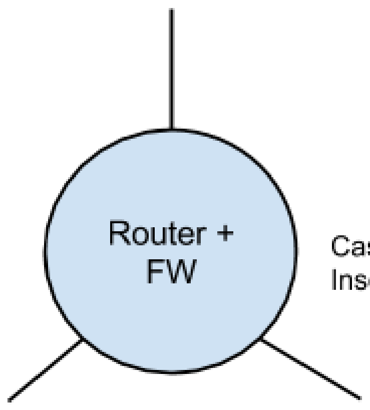
Key	Value	Default Value
networks	List of Uuids	None
subnets	List of Uuids	None
routers	List of uuids	None
ports	List off uuids	None
tap	boolean	False
in_chain	boolean	False

Based on the subnet, router and/or tap values, the provider could insert in one of the following ways (note that the insertion mode is inferred by the provider based on the context and does not have to be provided by the user):

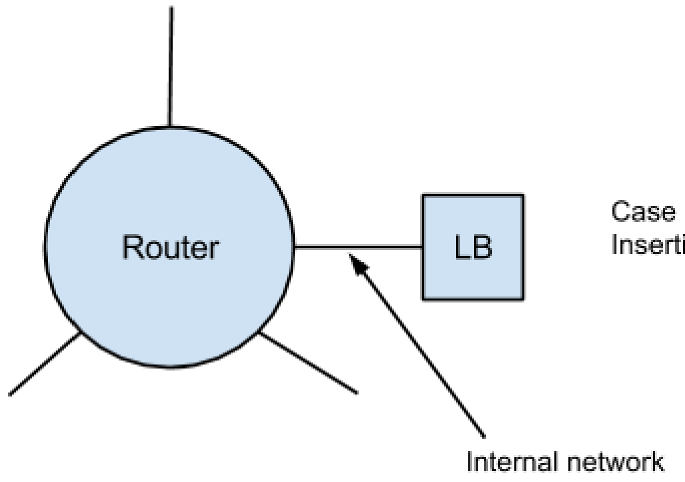
1. L3
   This service has an IP addressable point in the data path. It could be collocated with a router or is capable of L3-forwarding. Traffic to and from this service is directed by virtue of the IP address of the service and the L3 protocols in place. E.g. A Loadbalancer service.

The table below captures the different combinations of the routers, subnets, and tap attribute values (as provided by the users) and what insertion mode they correspond to (as inferred by the provider):

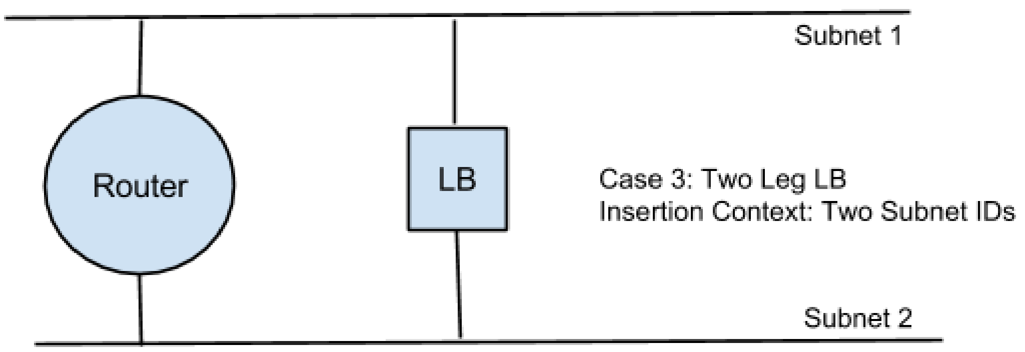
Routers (uuids)	Subnets (uuids)	Tap	Examples
1	None	False	Firewall is collocated with the router
1	None	False	One Arm loadbalancer. The difference between this and the above case is that the provider puts the loadbalancer on an internal network (not visible to the tenant) which is uplinked to the router.
None	>= 1	False	Loadbalancer with two legs (one leg in the VIP subnet, and another in the Pool subnet)
1	>=1	False	Loadbalancer in Direct Server Return (DSR) mode

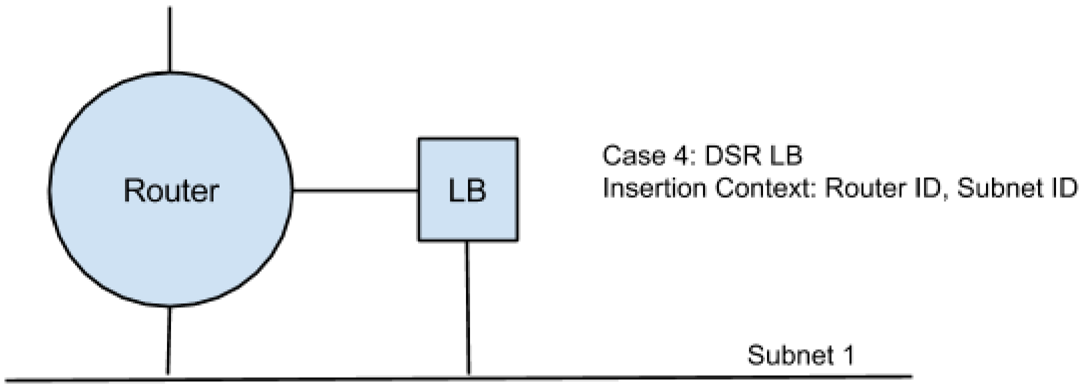


Case 1: FW is collocated with the Router
Insertion Context: Router ID



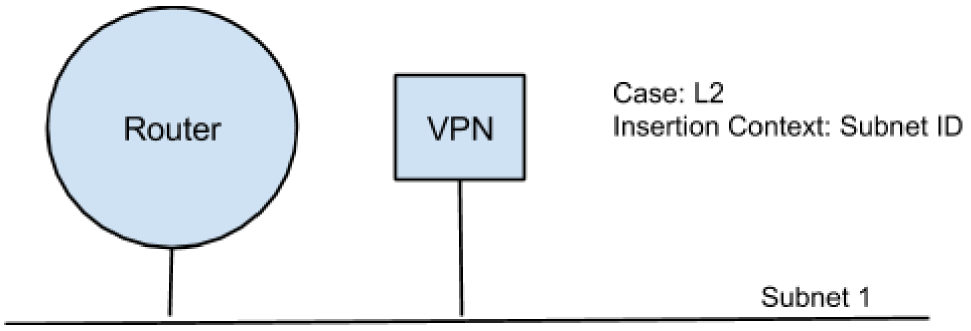
Case 2: One Arm LB
Insertion Context: Router ID





2. L2
   This service is capable of L2-switching and learns MAC addresses on its interfaces. The service switches traffic based on the MAC addresses it learns on it's ports. E.g. A L2-VPN service.

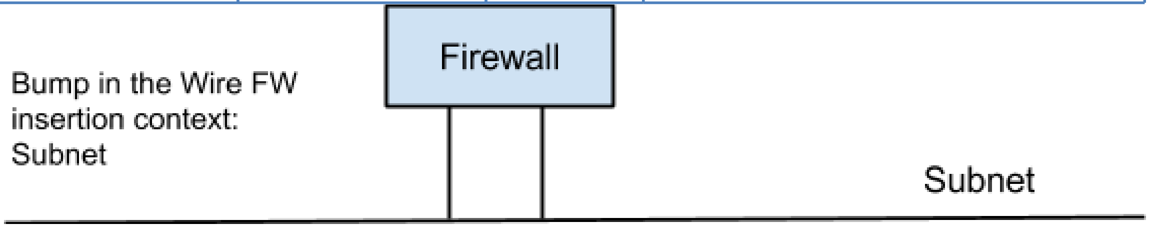
Routers (uuids)	Subnets (uuids)	Tap	Examples
None	1	False	L2 VPN



3. Bump-in-the-wire (BITW)

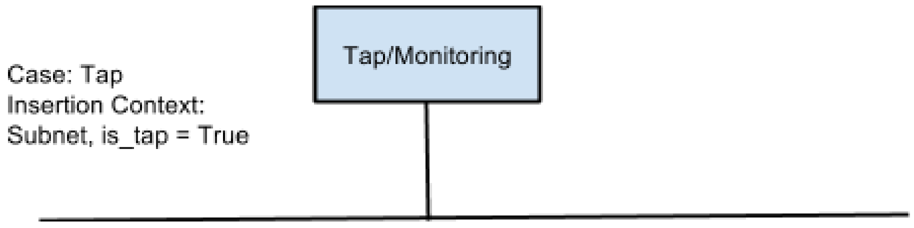
This service will not switch or route traffic and will typically have an ingress and an egress port. To insert this service, the traffic will have to be steered to the ingress port of the service. E.g. Firewall performing filtering and auditing.

Routers (uuids)	Subnets (uuids)	Tap	Examples
None	1	False	BITW Firewall



4. Tap
   This service only consumes the traffic at a particular point in the network topology. The traffic proceeds unaltered to it's destination. E.g. A monitoring service.

Routers (uuids)	Subnets (uuids)	Tap	Examples
None	1	True	Monitoring service



Note that the user ~~creates this resource~~ provides this service_context attribute when an individual service needs to be inserted. When the user requests that the service be a part of a service chain, the user provides the insertion context for the chain, and the plugin constructs and populates a relevant service_insertion_context for each service.

2. Service Chain (resource name: service_chains)

The service_chain resource is owned and managed by the tenant that creates it.

Attribute Name	Type	Default Value	Description
id	uuid-str	Generated	UUID of the service_chain.
tenant_id	uuid-str	N/A	Owner of the

			service_chain.
name	String	None	Human readable name for the service_chain.
description	String	None	
source_context	Dict	None	See definition of service_context earlier.
destination_context	Dict	None	See definition of service_context earlier.
service_graph	List	empty	List of service_uuids corresponding to services instantiated by the tenant.

To make the application of the service chain explicit, at least one of source or destination context need to be provided (both can be provided to make it completely explicit).

Source and destination can be the same (this is valid only for 'bump-in-the-wire' services).

The model makes use of the existing notion of an "external network" to define service chains that insert services in the path of traffic entering and exiting the tenant's network topology. The "external network" is created and published by the provider. Either source or destination could be an "external network".

Ideally, the source and destination would be group/classifier/policy constructs (or a combination of those) that capture application characteristics. However such constructs do not exist in Neutron and are outside the scope of this document. These can easily evolve as a parallel effort. (Update: a few blueprints have started to emerge in this area: https://blueprints.launchpad.net/neutron/+spec/policy-extensions-for-neutron, https://blueprints.launchpad.net/neutron/+spec/group-based-policy-abstraction)

4. Other changes to Existing Resources

~~Each service instance would need to have a reference to a `service_insertion_context` (this could be `None` for default insertion). This association can be introduced in at least a couple of different ways:~~

~~1. Add a service_insertion_context_id to each service_instance resource, or,~~

~~2. Introduce a ServiceInsertionContextResourceAssociation entity (similar to the ProviderResourceAssociation entity).~~

The existing service_providers resource can be augmented to support service_chain_providers which store the following tuples:
chain_name, list of service names from defined service_providers, provider module

5. API Operations

Base URL: /v2.0/

1. API at a glance

Service Chains		
GET	`/service_chains`	List summary of all service_chains created by this tenant
GET	`/service_chains/{service_chain_id}`	List the details of a particular service_chain
POST	`/service_chains`	Creates a new service_chain. Attributes in request body.
PUT	`/service_chains/{service_chain_id}`	Update a service_chain. Attributes in request body.
DELETE	`/service_chains/{service_chain_id}`	Delete the service_chain
~~Services Insertion Context~~		

~~GET~~	~~/service_insertion_contexts~~	~~List summary of all service_insertion_contexts created by this tenant~~
~~GET~~	~~/service_insertion_contexts/{service_insertion_context_id}~~	~~List the details of a particular service_insertion_context~~
~~POST~~	~~/service_insertion_contexts~~	~~Creates a new service_insertrion_context. Attributes in request body.~~
~~PUT~~	~~/service_insertion_contexts/{service_insertion_context_id}~~	~~Update a service_insertion_context. Attributes in request body.~~
~~DELETE~~	~~/service_insertion_contexts/{service_insertion_context_id}~~	~~Delete the service_insertion_context~~

6. Workflows (with CLI examples)

Admin first populates the supported providers:

This is the prevailing mechanism (see https://wiki.openstack.org/wiki/Neutron/ServiceTypeFramework for more details), the only addition here is the SERVICE_CHAIN service_type.

```
[service_providers]
#this is the existing format, no changes
service_provider=FIREWALL:IPTables:neutron.services.firewall.drivers.linux.i
ptables_fwaas.IptablesFwaasDriver:default
service_provider=FIREWALL:VendorA:neutron.services.firewall.drivers.vendorA.
FwaasDriver
service_provider=VPN:OpenSwan:neutron.services.vpn.drivers.OpenSwan:default
service_provider=VPN:VendorB:neutron.services.vpn.drivers.vendorB
```

```
[service_chain_providers]
#format:
#service_chain_provider=<chain_name>:<list of service names from 
above>:<provider module>

service_chain_provider=Firewall-VPN-Ref-Chain:[IPTables, 
OpenSwan]:neutron.services.chains.plugin
```

Create and insert individual service

1. Chooses Service Provider Name from list of available service providers.

```
neutron service-provider-list
-----------------------------------------------------
| Service Type        |         Name        | Default  |
|-----------------------------------------------------
| FIREWALL            |     IPTables        |  True    |
| FIREWALL            |     VendorA         |  False   |
| VPN                 |     OpenSwan        |  True    |
| VPN                 |     VendorB         |  False   |
-----------------------------------------------------
```

2. Create service instance (firewall in this case).
Default provider and default insertion context:

```
neutron firewall-create <firewall_policy_id>
```

OR
Specify provider and default insertion context:

```
neutron firewall-create <firewall_policy_id> --provider VendorA
```

OR
Specify provider and insertion context:

```
neutron firewall-create <firewall_policy_id> \
  --provider VendorA  \
  --service-context router_id=<router_id>
```

Create a Service Chain

1. Chooses Service Provider Name from list of available service providers.

```
neutron service-provider-list
-----------------------------------------------------
| Service Type        |         Name        | Default  |
|----------------------------------------------------
| FIREWALL            |     IPTables        |  True    |
| FIREWALL            |     VendorA         |  False   |
| VPN                 |     OpenSwan        |  True    |
| VPN                 |     VendorB         |  False   |
-----------------------------------------------------

neutron service-chain-provider-list

-----------------------------------------------
| Chain Name           |     Services         |
|----------------------------------------------
|Firewall-VPN-Ref-Chain| [IPTables,OpenSwan]  |
-----------------------------------------------
```

2. Create service instances (firewall and VPN in this case) without insertion context but by choosing the provider for the service chain.

```
neutron firewall-create <firewall_policy_id> \
  --provider IPTables  \
  --in-chain True      \
  … … …

neutron vpn-service-create \
  --provider OpenSwan  \
  --in-chain True      \
  … … …
```

3. Create service chain with insertion_context:

```
neutron service-chain-create \
  --provider Firewall-VPN-Ref-Chain  \
  --services <firewall_instance_id, vpn_instance_id> \
  --name my_fw_vpn_chain  \
  --source-insertion-context --router_id=<router_id>
  … … …
```

7. Use Cases

1. Bump-in-the wire L2 services

This is covered earlier.

2. L3 services collocated with the Router

This is covered earlier.

3. Services on Physical Hardware on Provider's network

The model is agnostic of this, but might require a provider workflow to introduce the presence of the service on the physical device. A proposal has been submitted to facilitate this: https://blueprints.launchpad.net/neutron/+spec/neutron-switch-port-extension

4. Services running on Tenant's Virtual Machines

This is covered in: https://blueprints.launchpad.net/neutron/+spec/adv-services-in-vms

5. One-arm Loadbalancer service

This is covered earlier.

6. Three/multitier Application Topologies

This is a combination of above use cases.

8. Implementation

- The following is an older implementation prior to the H release: https://github.com/bigswitch/quantum/tree/sumit/chaining