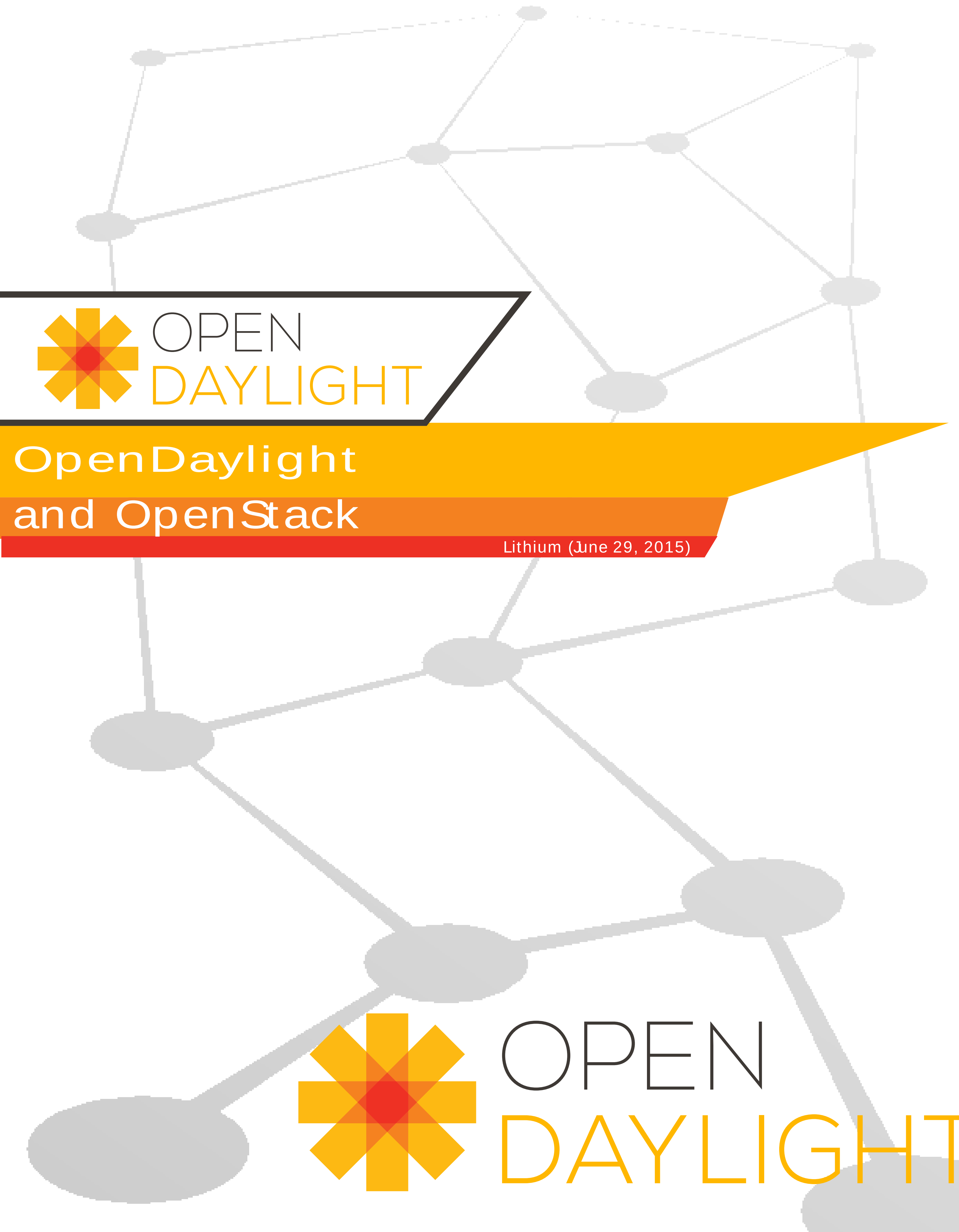


OpenDaylight and OpenStack

Lithium (June 29, 2015)

OpenDaylight and OpenStack

OpenDaylight Community

Lithium (2015-06-29)

This guide describes how to use OpenDaylight with OpenStack.

Table of Contents

List of Figures

Overview

OpenStack is a popular open source Infrastructure as a service project, covering compute, storage and network management. OpenStack can use OpenDaylight as its network management provider through the Modular Layer 2 (ML2) north-bound plug-in. OpenDaylight manages the network flows for the OpenStack compute nodes via the OVSDB south-bound plug-in. This page describes how to set that up, and how to tell when everything is working.

1. Installing OpenStack

Installing OpenStack is out of scope for this document, but to get started, it is useful to have a minimal multi-node OpenStack deployment.

The reference deployment we will use for this document is a 3 node cluster:

- One control node containing all of the management services for OpenStack (Nova, Neutron, Glance, Swift, Cinder, Keystone)
- Two compute nodes running nova-compute
- Neutron using the OVS back-end and vxlan for tunnels

Once you have installed OpenStack, verify that it is working by connecting to Horizon and performing a few operations. To check the Neutron configuration, create two instances on a private subnet bridging to your public network, and verify that you can connect to them, and that they can see each other.

2. Installing OpenDaylight

Table of Contents

OpenStack with OVSDB

Prerequisites: OpenDaylight requires Java 1.7.0.

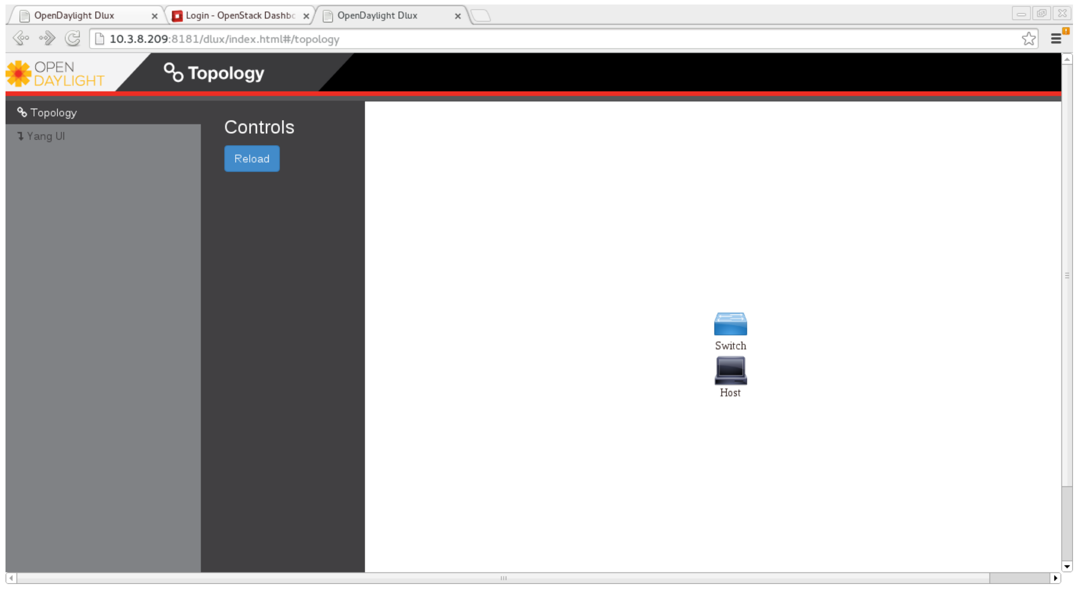
- On the control host, Download the latest OpenDaylight release (at the time of writing, this is 0.2.1-Helium-SR1.1)
- Uncompress it as root, and start OpenDaylight (you can start OpenDaylight by running karaf directly, but exiting from the shell will shut it down):

```
$ tar xvfz distribution-karaf-0.2.1-Helium-SR1.1.tar.gz
$ cd distribution-karaf-0.2.0-Helium
$ ./bin/start # Start OpenDaylight as a server process
```

- Connect to the Karaf shell, and install the odl-ovsdb-openstack bundle, dlux and their dependencies:

```
$ ./bin/client # Connect to OpenDaylight with the client
opendaylight-user@root> feature:install odl-base-all odl-aaa-authn odl-
restconf odl-nsf-all odl-adsal-northbound odl-mdsal-apidocs \
odl-ovsdb-openstack odl-ovsdb-northbound odl-dlux-core
```

- If everything is installed correctly, you should now be able to log in to the dlux interface on `http://$CONTROL_HOST:8181/dlux/index.html` - the default username and password is "admin/admin" (see screenshot below)



Ensuring OpenStack network state is clean

When using OpenDaylight as the Neutron back-end, ODL expects to be the only source of truth for Open vSwitch configuration. Because of this, it is necessary to remove existing OpenStack and Open vSwitch configurations to give OpenDaylight a clean slate.

- Delete instances

```
$ nova list
$ nova delete <instance names>
```

- Remove link from subnets to routers

```
$ neutron subnet-list
$ neutron router-list
$ neutron router-port-list <router name>
$ neutron router-interface-delete <router name> <subnet ID or name>
```

- Delete subnets, nets, routers

```
$ neutron subnet-delete <subnet name>
$ neutron net-list
$ neutron net-delete <net name>
$ neutron router-delete <router name>
```

- Check that all ports have been cleared - at this point, this should be an empty list

```
$ neutron port-list
```

Ensure Neutron is stopped

While Neutron is managing the OVS instances on compute and control nodes, OpenDaylight and Neutron can be in conflict. To prevent issues, we turn off Neutron server on the network controller, and Neutron's Open vSwitch agents on all hosts.

- Turn off neutron-server on control node

```
# systemctl stop neutron-server
```

- On each node in the cluster, shut down and disable Neutron's agent services to ensure that they do not restart after a reboot:

```
# systemctl stop neutron-openvswitch-agent
# systemctl disable neutron-openvswitch-agent
```

Configuring Open vSwitch to be managed by OpenDaylight

On each host (both compute and control nodes) we will clear the pre-existing Open vSwitch config and set OpenDaylight to manage the switch:

- Stop the Open vSwitch service, and clear existing OVSDB (ODL expects to manage vSwitches completely)

```
# systemctl stop openvswitch
# rm -rf /var/log/openvswitch/*
# rm -rf /etc/openvswitch/conf.db
# systemctl start openvswitch
```

- At this stage, your Open vSwitch configuration should be empty:

```
[root@dneary-odl-compute2 ~]# ovs-vsctl show
9f3b38cb-eefc-4bc7-828b-084b1f66fbfd
    ovs_version: "2.1.3"
```

- Set OpenDaylight as the manager on all nodes

```
# ovs-vsctl set-manager tcp:${CONTROL_HOST}:6640
```

- You should now see a new section in your Open vSwitch configuration showing that you are connected to the OpenDaylight server, and OpenDaylight will automatically create a br-int bridge:

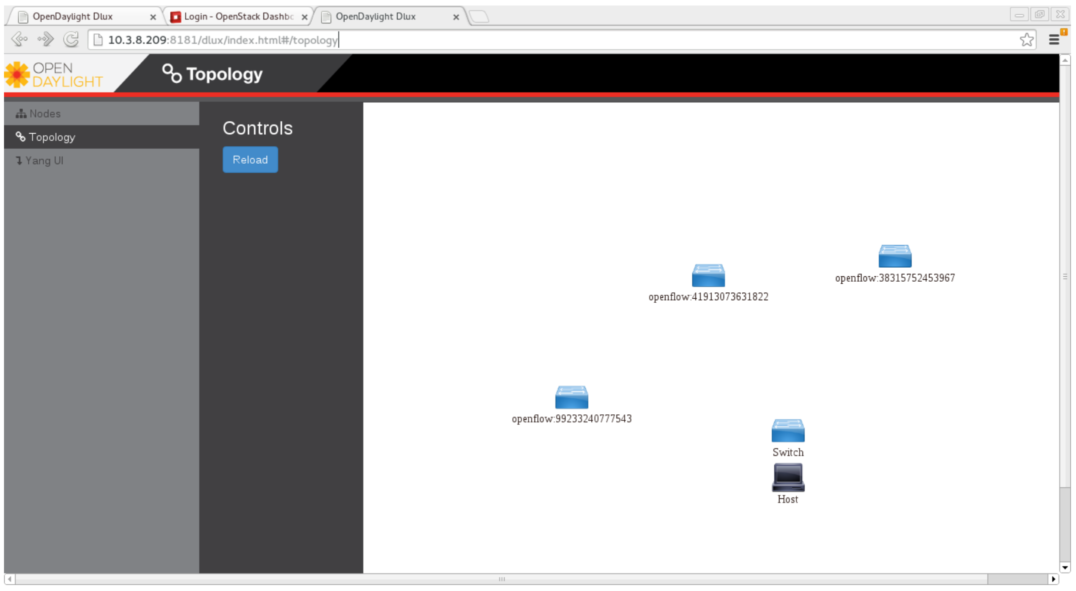
```
[root@dneary-odl-compute2 ~]# ovs-vsctl show
9f3b38cb-eefc-4bc7-828b-084b1f66fbfd
    Manager "tcp:172.16.21.56:6640"
        is_connected: true
    Bridge br-int
        Controller "tcp:172.16.21.56:6633"
        fail_mode: secure
        Port br-int
            Interface br-int
    ovs_version: "2.1.3"
```

- (BUG WORKAROUND) If SELinux is enabled, you may not have a security context in place which allows Open vSwitch remote administration. If you do not see the result above (specifically, if you do not see "is_connected: true" in the Manager section), set SELinux to Permissive mode on all nodes and ensure it stays that way after boot:

```
# setenforce 0
# sed -i -e 's/SELINUX=enforcing/SELINUX=permissive/g' /etc/selinux/config
```

- Make sure all nodes, including the control node, are connected to OpenDaylight

- If you reload DLUX, you should now see that all of your Open vSwitch nodes are now connected to OpenDaylight



- If something has gone wrong, check <code>data/log/karaf.log</code> under the OpenDaylight distribution directory. If you do not see any interesting log entries, set logging for OVSDB to TRACE level inside Karaf and try again:

```
log:set TRACE ovsdb
```

Configuring Neutron to use OpenDaylight

Once you have configured the vSwitches to connect to OpenDaylight, you can now ensure that OpenStack Neutron is using OpenDaylight.

First, ensure that port 8080 (which will be used by OpenDaylight to listen for REST calls) is available. By default, swift-proxy-service listens on the same port, and you may need to move it (to another port or another host), or disable that service. I moved it to port 8081 by editing <code>/etc/swift/proxy-server.conf</code> and <code>/etc/cinder/cinder.conf</code>, modifying iptables appropriately, and restarting swift-proxy-service and OpenDaylight.

- Configure Neutron to use OpenDaylight's ML2 driver:

```
crudini --set /etc/neutron/plugins/ml2/ml2_conf.ini ml2 mechanism_drivers
 opendaylight
crudini --set /etc/neutron/plugins/ml2/ml2_conf.ini ml2 tenant_network_types
 vxlan

cat <<EOT>> /etc/neutron/plugins/ml2/ml2_conf.ini
[ml2_odl]
password = admin
username = admin
url = http://${CONTROL_HOST}:8080/controller/nb/v2/neutron
EOT
```

- Reset Neutron's ML2 database

```
mysql -e "drop database if exists neutron_ml2;"
mysql -e "create database neutron_ml2 character set utf8;"
mysql -e "grant all on neutron_ml2.* to 'neutron'@'%';"
neutron-db-manage --config-file /usr/share/neutron/neutron-dist.conf --
config-file /etc/neutron/neutron.conf \
--config-file /etc/neutron/plugin.ini upgrade head
```

- Restart neutron-server:

```
systemctl start neutron-server
```

Verifying it works

- Verify that OpenDaylight's ML2 interface is working:

```
curl -u admin:admin http://${CONTROL_HOST}:8080/controller/nb/v2/neutron/
networks

{
   "networks" : [ ]
}
```

If this does not work or gives an error, check Neutron's log file in <code>/var/log/neutron/server.log</code>. Error messages here should give some clue as to what the problem is in the connection with OpenDaylight

- Create a net, subnet, router, connect ports, and start an instance using the Neutron CLI:

  ```
  neutron router-create router1
  neutron net-create private
  neutron subnet-create private --name=private_subnet 10.10.5.0/24
  neutron router-interface-add router1 private_subnet
  nova boot --flavor <flavor> --image <image id> --nic net-id=<network id>
   test1
  nova boot --flavor <flavor> --image <image id> --nic net-id=<network id>
   test2
  ```

  At this point, you have confirmed that OpenDaylight is creating network end-points for instances on your network and managing traffic to them.

Congratulations! You're done!

OpenStack with GroupBasedPolicy

This section is for Application Developers and Network Administrators who are looking to integrate Group Based Policy with OpenStack.

To enable the **GBP** Neutron Mapper feature, at the karaf console:

```
feature:install odl-groupbasedpolicy-neutronmapper
```

Neutron Mapper has the following dependencies that are automatically loaded:

```
odl-neutron-service
```

Neutron Northbound implementing REST API used by OpenStack

```
odl-groupbasedpolicy-base
```

Base **GBP** feature set, such as policy resolution, data model etc.

```
odl-groupbasedpolicy-ofoverlay
```

For Lithium, **GBP** has one renderer, hence this is loaded by default.

REST calls from OpenStack Neutron are by the Neutron NorthBound project.

GBP provides the implementation of the Neutron V2.0 API.

Features

List of supported Neutron entities:

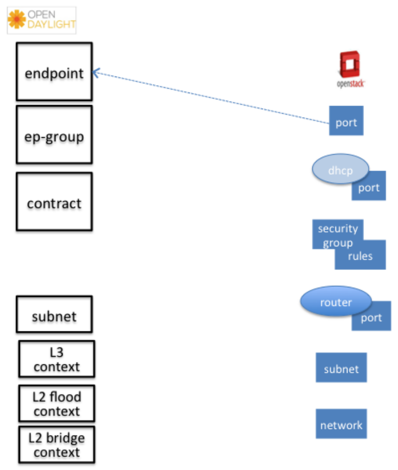
- Port
- Network
  - Standard Internal

  - External provider L2/L3 network
- Subnet
- Security-groups
- Routers
  - Distributed functionality with local routing per compute
  - External gateway access per compute node (dedicated port required)
  - Multiple routers per tenant
- FloatingIP NAT
- IPv4/IPv6 support

The mapping of Neutron entities to **GBP** entities is as follows:

Neutron Port

Figure 2.1. Neutron Port



The Neutron port is mapped to an endpoint.

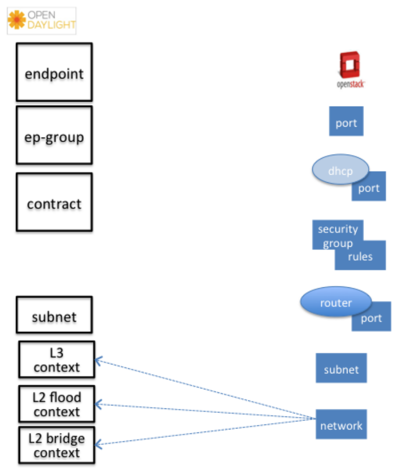
The current implementation supports one IP address per Neutron port.

An endpoint and L3-endpoint belong to multiple EndpointGroups if the Neutron port is in multiple Neutron Security Groups.

The key for endpoint is L2-bridge-domain obtained as the parent of L2-flood-domain representing Neutron network. The MAC address is from the Neutron port. An L3-endpoint is created based on L3-context (the parent of the L2-bridge-domain) and IP address of Neutron Port.

Neutron Network

Figure 2.2. Neutron Network



A Neutron network has the following characteristics:

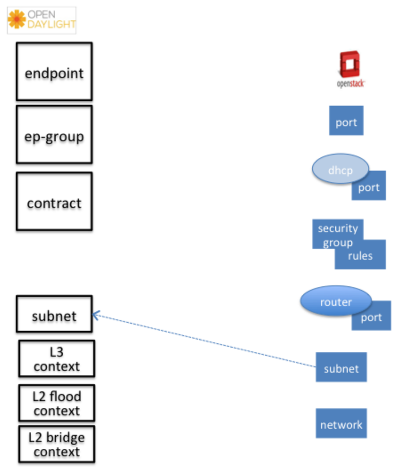
- defines a broadcast domain
- defines a L2 transmission domain
- defines a L2 name space.

To represent this, a Neutron Network is mapped to multiple **GBP** entities. The first mapping is to an L2 flood-domain to reflect that the Neutron network is one flooding or broadcast domain. An L2-bridge-domain is then associated as the parent of L2 flood-domain. This reflects both the L2 transmission domain as well as the L2 addressing namespace.

The third mapping is to L3-context, which represents the distinct L3 address space. The L3-context is the parent of L2-bridge-domain.

Neutron Subnet

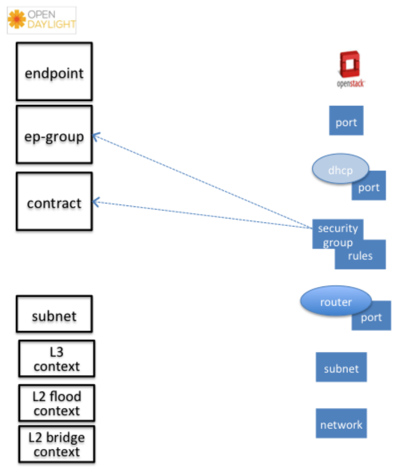
Figure 2.3. Neutron Subnet



Neutron subnet is associated with a Neutron network. The Neutron subnet is mapped to a **GBP** subnet where the parent of the subnet is L2-flood-domain representing the Neutron network.

Neutron Security Group

Figure 2.4. Neutron Security Group and Rules



GBP entity representing Neutron security-group is EndpointGroup.

Infrastructure EndpointGroups

Neutron-mapper automatically creates EndpointGroups to manage key infrastructure items such as:

- DHCP EndpointGroup - contains endpoints representing Neutron DHCP ports
- Router EndpointGroup - contains endpoints representing Neutron router interfaces
- External EndpointGroup - holds L3-endpoints representing Neutron router gateway ports, also associated with FloatingIP ports.

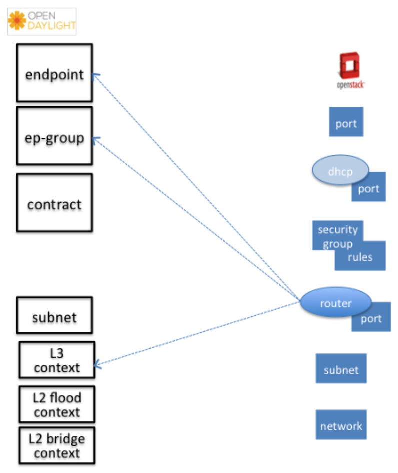
Neutron Security Group Rules

This mapping is most complicated among all others because Neutron security-group-rules are mapped to contracts with clauses, subjects, rules, action-refs, classifier-refs, etc. Contracts are used between endpoint groups representing Neutron Security Groups. For simplification it is important to note that Neutron security-group-rules are similar to a **GBP** rule containing:

- classifier with direction
- action of **allow**.

Neutron Routers

Figure 2.5. Neutron Router



Neutron router is represented as a L3-context. This treats a router as a Layer3 namespace, and hence every network attached to it a part of that Layer3 namespace.

This allows for multiple routers per tenant with complete isolation.

The mapping of the router to an endpoint represents the router's interface or gateway port.

The mapping to an EndpointGroup represents the internal infrastructure EndpointGroups created by the **GBP** Neutron Mapper

When a Neutron router interface is attached to a network/subnet, that network/subnet and its associated endpoints or Neutron Ports are seamlessly added to the namespace.

Neutron FloatingIP

When associated with a Neutron Port, this leverages the **GBP** OfOverlay renderer's NAT capabilities.

A dedicated *external* interface on each Nova compute host allows for disitributed external access. Each Nova instance associated with a FloatingIP address can access the external network directly without having to route via the Neutron controller, or having to enable any form of Neutron distributed routing functionality.

Assuming the gateway provisioned in the Neutron Subnet command for the external network is reachable, the combination of **GBP** Neutron Mapper and OfOverlay renderer will automatically ARP for this default gateway, requiring no user intervention.

Troubleshooting within GBP

Logging level for the mapping functionality can be set for package org.opendaylight.groupbasedpolicy.neutron.mapper. An example of enabling TRACE logging level on karaf console:

```
log:set TRACE org.opendaylight.groupbasedpolicy.neutron.mapper
```

Neutron mapping example

As an example for mapping can be used creation of Neutron network, subnet and port. When a Neutron network is created 3 **GBP** entities are created: l2-flood-domain, l2-bridge-domain, l3-context.

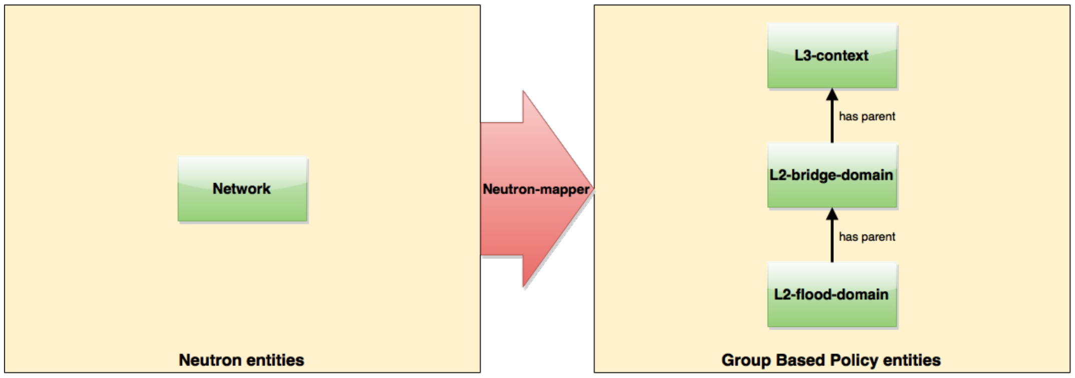


Figure 2.6. Neutron network mapping

After an subnet is created in the network mapping looks like this.

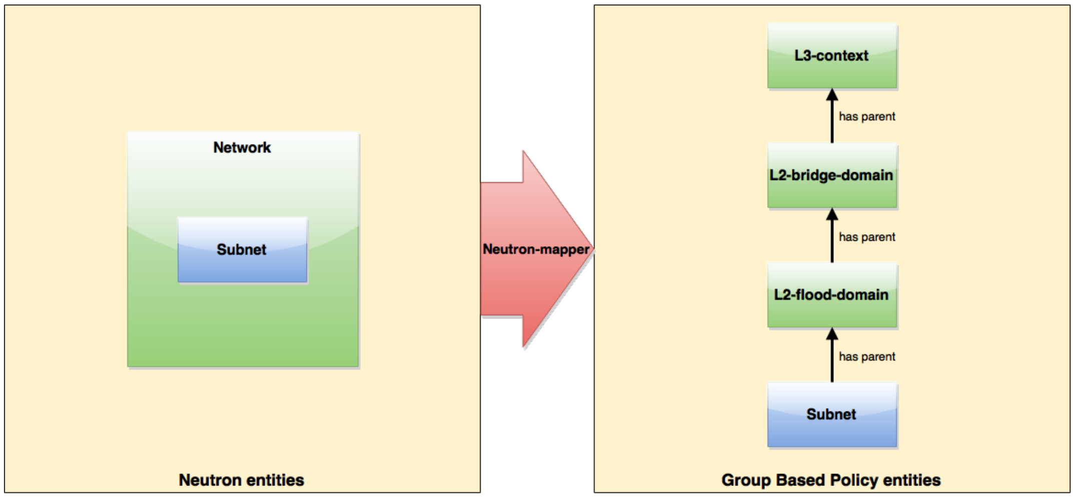
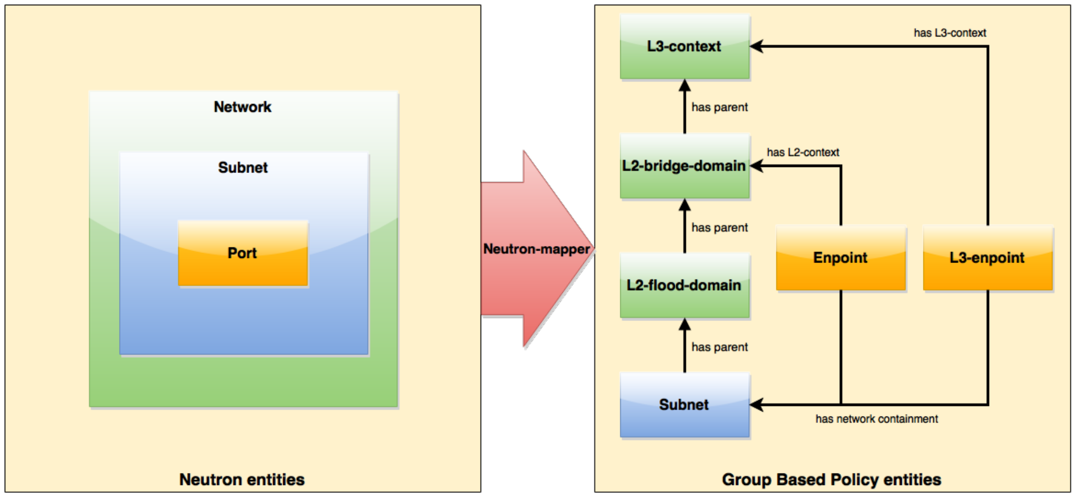


Figure 2.7. Neutron subnet mapping

If an Neutron port is created in the subnet an endpoint and l3-endpoint are created. The endpoint has key composed from l2-bridge-domain and MAC address from Neutron port. A key of l3-endpoint is compesed from l3-context and IP address. The network containment of endpoint and l3-endpoint points to the subnet.

Figure 2.8. Neutron port mapping



Configuring GBP Neutron

No intervention passed initial OpenStack setup is required by the user.

More information about configuration can be found in our DevStack demo environment on the **GBP** wiki.

Administering or Managing GBP Neutron

For consistencies sake, all provisioning should be performed via the Neutron API. (CLI or Horizon).

The mapped policies can be augmented via the **GBP** UX,UX, to:

- Enable Service Function Chaining
- Add endpoints from outside of Neutron i.e. VMs/containers not provisioned in OpenStack
- Augment policies/contracts derived from Security Group Rules
- Overlay additional contracts or groupings

Tutorials

A DevStack demo environment can be found on the **GBP** wiki.