# **How metadata service work by default (in router namespace)**

查看neuron相关信息：

[root@compute1 ~]# ip netns

**qrouter-05ad9fe0-c4ef-4843-ac5d-7c29faa69721 (id: 10)**

**qrouter-d69f2225-882b-4b3f-8e99-4e1b0c4675c5 (id: 9)**

qdhcp-630c08ec-2c84-4875-8946-06d158bdf57e (id: 8)

qdhcp-f860ff64-4f1d-4e37-a075-58a5f1d3c57f (id: 7)

qdhcp-e92a7f1b-879e-4435-baa2-1c52d04d2b66 (id: 6)

qdhcp-d2ff7bca-33b2-4700-be2f-e45c64e99e50 (id: 5)

qdhcp-bc029c34-1a3d-4d97-a320-2e7a9c641b2f (id: 4)

qdhcp-287df61d-e8ee-4937-825f-6d996c136e58 (id: 2)

qdhcp-d50dd8ad-6789-425f-9cc9-db4d6d05c1a6 (id: 3)

qdhcp-1be3a757-5adb-4263-8eda-14ed2b4cb30b (id: 1)

qdhcp-0a2d26a2-b223-437d-af42-5ae04b972449 (id: 0)

[root@compute1 ~]#

通过命令，查看9697端口信息

[root@compute1 ~]# ip netns exec **qrouter-d69f2225-882b-4b3f-8e99-4e1b0c4675c5** netstat -ntlp

Active Internet connections (only servers)

Proto Recv-Q Send-Q Local Address Foreign Address State PID/Program name

tcp 0 0 0.0.0.0:9697 0.0.0.0:\* LISTEN **8891**/python2

[root@compute1 ~]# ip netns exec qrouter-05ad9fe0-c4ef-4843-ac5d-7c29faa69721 netstat -ntlp

Active Internet connections (only servers)

Proto Recv-Q Send-Q Local Address Foreign Address State PID/Program name

tcp 0 0 0.0.0.0:9697 0.0.0.0:\* LISTEN 8889/python2

[root@compute1 ~]# ps -f --pid 8891 |grep fold -s -w 100

[root@compute1 ~]# ps -f --pid **8891** | fold -s -w 100

UID PID PPID C STIME TTY TIME CMD

neutron 8891 1 0 09:26 ? 00:00:01 /usr/bin/python2 /bin/neutron-ns-metadata-proxy

--pid\_file=/var/lib/neutron/external/pids/d69f2225-882b-4b3f-8e99-4e1b0c4675c5.pid

--metadata\_proxy\_socket=/var/lib/neutron/metadata\_proxy

--router\_id=d69f2225-882b-4b3f-8e99-4e1b0c4675c5 --state\_path=/var/lib/neutron --metadata\_port=**9697**

--metadata\_proxy\_user=995 --metadata\_proxy\_group=992 --debug --verbose

--log-file=neutron-ns-metadata-proxy-d69f2225-882b-4b3f-8e99-4e1b0c4675c5.log

--log-dir=/var/log/neutron

[root@compute1 ~]#

iptables rule to redirect metadata request sent to **http://169.254.169.254:80** to **9697** port：

[root@compute1 ~]# ip netns exec qrouter-d69f2225-882b-4b3f-8e99-4e1b0c4675c5 iptables-save | grep REDIRECT

-A neutron-l3-agent-PREROUTING -d 169.254.169.254/32 -i qr-+ -p tcp -m tcp --dport 80 -j REDIRECT --to-ports 9697

[root@compute1 ~]#

# **Metadata service in DHCP namespace**

获取neutron的dhcp信息：

[root@compute1 ~]# ip netns

qrouter-05ad9fe0-c4ef-4843-ac5d-7c29faa69721 (id: 10)

qrouter-d69f2225-882b-4b3f-8e99-4e1b0c4675c5 (id: 9)

**qdhcp-630c08ec-2c84-4875-8946-06d158bdf57e** (id: 8)

qdhcp-f860ff64-4f1d-4e37-a075-58a5f1d3c57f (id: 7)

qdhcp-e92a7f1b-879e-4435-baa2-1c52d04d2b66 (id: 6)

qdhcp-d2ff7bca-33b2-4700-be2f-e45c64e99e50 (id: 5)

qdhcp-bc029c34-1a3d-4d97-a320-2e7a9c641b2f (id: 4)

qdhcp-287df61d-e8ee-4937-825f-6d996c136e58 (id: 2)

qdhcp-d50dd8ad-6789-425f-9cc9-db4d6d05c1a6 (id: 3)

qdhcp-1be3a757-5adb-4263-8eda-14ed2b4cb30b (id: 1)

qdhcp-0a2d26a2-b223-437d-af42-5ae04b972449 (id: 0)

[root@compute1 ~]#

[root@compute1 ~]# ip netns exec **qdhcp-630c08ec-2c84-4875-8946-06d158bdf57e** ip -4 a

1: lo: <LOOPBACK,UP,LOWER\_UP> mtu 65536 qdisc noqueue state UNKNOWN qlen 1

inet 127.0.0.1/8 scope host lo

valid\_lft forever preferred\_lft forever

2: ns-80f5d4db-96@if40: <BROADCAST,MULTICAST,UP,LOWER\_UP> mtu 1500 qdisc noqueue state UP qlen 1000 link-netnsid 0

inet 10.12.15.236/24 brd 10.12.15.255 scope global ns-80f5d4db-96

valid\_lft forever preferred\_lft forever

inet **169.254.169.254/16** brd 169.254.255.255 scope global ns-80f5d4db-96

valid\_lft forever preferred\_lft forever

[root@compute1 ~]#

# Identifying and Troubleshooting Neutron Namespaces

[Damian Igbe](https://www.mirantis.com/author/damian-igbe/) on November 21, 2013

I [recently](https://www.mirantis.com/blog/the-road-to-hong-kong-openstack-summit-speakers-4-neutron-network-namespaces-and-iptables/) gave you an intro to my talk at the Hong Kong OpenStack Summit about Neutron namespaces. One of the commenters had seen the video of my talk and requested that we post the slideshow. In this post, I will show you how to:

Identify the correct namespace.

Perform general troubleshooting around the identified namespace.

And, I will also share my presentation from that talk at the end of the post.

Let’s dive right in. If you run both L3 + DHCP services on the same node, you should enable namespaces to avoid conflicts with routes:

Use\_namespaces=True

To disable namespaces, make sure the neutron.conf used by the neutron-server has the following setting:

allow\_overlapping\_ips=False

And that the dhcp\_agent.ini and l3\_agent.ini have the following setting:

use\_namespaces=False

How are namespaces implemented in Neutron and how do we recognize them? Let us begin by stating the following 2 very important facts:

1,Every l2-agent/private network has an associated dhcp namespace and

2.Every l3-agent/router has an associated router namespace.

OK, but how are namespaces named? A network namespace starts with dhcp-followed by the ID of the network. Similarly, a router namespace starts with qrouter- followed by the ID of the router. For example:

stack@vmnet-mn:~$ sudo ip netns list

qdhcp-eb2367bd-6e43-4de7-a0ab-d58ebf6e7dc0

qrouter-4adef4a5-1122-49b8-9da2-3cbf803c44dd

The above namespace listing is for a simple network consisting of a single private network and a single router. To help our understanding of this simple network, let us use the following illustration in Figure 1

****Figure 1 Illustration of a simple network with router and DHCP namespaces****

Figure 1 illustrates how a tenant network is constructed with router and dhcp namespaces. A single virtual machine (VM1) is attached for illustration purposes. For each tenant, as networks are created, similar network topologies will be created. In figure 1, we make the following observations:

Data center network is the external network by which packets exit to the internet.

The router with IP address 10.0.0.1 connects the tenant network with the data center network? It has an associated router namespace so that the tenant routing tables are isolated from every other tenant in the OpenStack deployment.

The bridge represents the tenant network of 10.0.0.0/24. The DHCP server (here a dnsmasq process) is attached the private network and has an associated IP address of 10.0.0.2  just as a physical server  attached to the network. Note that the router usually takes the first IP address while the DHCP server takes the next IP address and the VMs take addresses from the remaining pool of IP addresses. Similarly, the DHCP server has an associated DHCP namespace so that the IP address allocation and all broadcast traffic are kept within this tenant network.

## Neutron namespace troubleshooting

As a fundamental concept in OpenStack Networking, without a proper understanding of network namespaces, network troubleshooting is like finding a needle in a haystack. Consider a small OpenStack deployment with about 1000 tenants and each tenant Neutron network consisting of a router and one private network at the minimum. This will make a total of at least 2000 namespaces because each tenant router is a namespace while each network will have an associated DHCP namespace (two namespaces per tenant, as illustrated in figure 1). In such a scenario, let’s say that a tenant is having issues with the DHCP allocation whereby the instance shows the assigned IP addresses when viewed through the OpenStack Dashboard, yet you cannot SSH into the instance and, looking through the VNC, the network interface appears not have an IP address.

Troubleshooting this problem will involve looking at various components of the tenant topology as shown in figure 1, primarily the DHCP agent. The first logical step is to identify the tenant namespaces. Now we can see how this is different from troubleshooting in a traditional physical network deployment. Here, without a proper understanding of how namespace work in Neutron, the administrator will be stuck and won’t be able to proceed with logical troubleshooting procedures.

Let us summarize the troubleshooting steps:

Identify the correct namespace

Perform general troubleshooting around the identified namespace

Let’s create two tenant networks and use them to illustrate troubleshooting namespaces.

****Figure 2 Network for tenant A****

****Figure 3 Network for tenant B****

## Step 1: Identify the namespace

In the above diagrams, network for Tenant A has 2 private networks and one router while network for tenant B also has 2 networks and a router. Let us explore these networks.  Note that in a multimode deployment consisting of a controller node, a network node and compute nodes, we will be running the namespace commands from the network node.

stack@vmnet-mn:~$ sudo ip netns list

[sudo] password for stack:

qdhcp-eb2367bd-6e43-4de7-a0ab-d58ebf6e7dc0

qdhcp-9f7ec6fd-ec9a-4b1b-b854-a9832f190922

qrouter-4587f8d0-08a7-434a-aaeb-74e68a50c48d

qdhcp-f5a69443-bf3f-4f9a-b3e2-5625e70514b4

qrouter-4adef4a5-1122-49b8-9da2-3cbf803c44dd

qdhcp-e0fe9037-790a-4c6b-9bf4-4b06f0cfcf5c

We can six namespaces listed from the two tenant networks (three namespaces per tenant).

OK, but how do you know which namespaces belong to which tenant since the tenant names are not indicated in the namespaces? We can approach this in two different ways.

First, note that typically, only the cloud administrator will have access to the network node and therefore only the administrator can run the namespaces commands on the network node. One way the cloud administrator could identify the namespaces is to source the credential file of a user belonging to the tenant in question. Once that is done, the entire neutron commands will run as that user and only the tenant network components where the user belongs to will display. Let us illustrate this as follows.

Source a credential file that belongs to a user in tenant A.

Run the following commands:

stack@vmnet-mn:~$ neutron router-list

+--------------------------------------+-----------------+--------------------------------------------------------+

| id | name | external\_gateway\_info |

+--------------------------------------+-----------------+--------------------------------------------------------+

| 4adef4a5-1122-49b8-9da2-3cbf803c44dd | router\_proj\_one | {"network\_id": "ea00b8c3-7063-4780-ad73-ef0b47518f10"} |

+--------------------------------------+-----------------+--------------------------------------------------------+

From the listing we can see that tenant A has one router listed with ID 4adef4a5-1122-49b8-9da2-3cbf803c44dd

Looking for this ID in the ip netns command above, we can easily identify namespace qrouter-4adef4a5-1122-49b8-9da2-3cbf803c44dd as belonging to tenant A.

For the dhcp namespaces:

stack@vmnet-mn:~$ neutron net-list

+--------------------------------------+--------------+-----------------------------------------------------+

| id | name | subnets |

+--------------------------------------+--------------+-----------------------------------------------------+

| e0fe9037-790a-4c6b-9bf4-4b06f0cfcf5c | net\_proj\_one | f02b6df4-11af-411d-b41e-e4ce66d5510d 10.10.1.0/24 |

| ea00b8c3-7063-4780-ad73-ef0b47518f10 | ext\_net | 4f7c6c9e-28d8-43b3-b7d4-276920216b31 |

| f5a69443-bf3f-4f9a-b3e2-5625e70514b4 | net2 | f6cd4df6-b37f-4615-94f6-8abcab38ef99 192.168.0.0/24 |

+--------------------------------------+--------------+-----------------------------------------------------+

Also in this listing we can identify two private networks with IDs e0fe9037-790a-4c6b-9bf4-4b06f0cfcf5c and f5a69443-bf3f-4f9a-b3e2-5625e70514b4.

Checking the ip netns list command above, we can also identify the corresponding dhcp namespaces as: qdhcp-e0fe9037-790a-4c6b-9bf4-4b06f0cfcf5c and qdhcp-f5a69443-bf3f-4f9a-b3e2-5625e70514b4

The above procedures are good and easiest if the cloud administrator knows the username/password of the tenant but this will not always be the case. So another way the cloud administrator could identify the tenant namespaces is to ask the user for the ID of the network or the name of the network. The tenant user can easily find out this information from Horizon, the OpenStack dashboard. With either the name or the ID, the cloud administrator can troubleshoot as follows:

Source admin credentials. With the admin credential, the cloud administrator will have access to every tenant network namespaces

1. List the namespaces and select the tenant namespace

stack@vmnet-mn:~$ ip netns list | grep 9f7ec6fd-ec9a-4b1b-b854-a9832f190922

qdhcp-9f7ec6fd-ec9a-4b1b-b854-a9832f190922

## Step 2: Troubleshooting around the namespace

Now that we know the namespace of the tenant we can troubleshoot around the namespace.

First, let us check the IP address.

stack@vmnet-mn:~$ sudo ip netns exec qdhcp-9f7ec6fd-ec9a-4b1b-b854-a9832f190922 ifconfig

[sudo] password for stack:

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:16 errors:0 dropped:0 overruns:0 frame:0

TX packets:16 errors:0 dropped:0 overruns:0 carrier:0

collisions:0 txqueuelen:0

RX bytes:1344 (1.3 KB) TX bytes:1344 (1.3 KB)

tap43cb2c73-40 Link encap:Ethernet HWaddr fa:16:3e:e6:1b:24

inet addr:10.1.0.3 Bcast:10.1.0.255 Mask:255.255.255.0

inet6 addr: fe80::f816:3eff:fee6:1b24/64 Scope:Link

UP BROADCAST RUNNING MULTICAST MTU:1500 Metric:1

RX packets:965 errors:0 dropped:0 overruns:0 frame:0

TX packets:561 errors:0 dropped:0 overruns:0 carrier:0

collisions:0 txqueuelen:0

RX bytes:178396 (178.3 KB) TX bytes:93900 (93.9 KB)

Let’s then ping the interface:

stack@vmnet-mn:~$ sudo ip netns exec qdhcp-9f7ec6fd-ec9a-4b1b-b854-a9832f190922 ping –c 3 10.1.0.3

PING 10.1.0.3 (10.1.0.3) 56(84) bytes of data.

64 bytes from 10.1.0.3: icmp\_req=1 ttl=64 time=0.081 ms

64 bytes from 10.1.0.3: icmp\_req=2 ttl=64 time=0.035 ms

64 bytes from 10.1.0.3: icmp\_req=3 ttl=64 time=0.055 ms

With the above steps we have identified the correct namespace and verify that it has the correct IP address that is pingable, and hence up and running. Further troubleshooting steps could involve the following:

* Check that the dnsmasq process attached to the subnet is upFor the DHCP, you can check the process

stack@vmnet-mn:~$ ps –aux | grep dhcp

* From the output of the above command, identify that there is a process with the network of choice (for example, 10.0.0.0) else the dnsmasq process attached to that network may not be running.
* Check that there is no firewall blocking communication to/from the subnet.

## Summary

Before we conclude, take note of the following:

* When a router or network is created, the namespaces don’t get created immediately. For network, the DHCP namespaces get created only when a vm is attached and for router the namespace is created when a gateway is set. It means that an activity must take place before the namespaces get created.
* When a router or network is deleted, the associated namespaces are not deleted.  They need to be manually deleted.

As Network namespaces are very important to OpenStack networking so is the understanding of network namespaces important to the [OpenStack cloud](https://www.mirantis.com/openstack-cloud/?utm_campaign=OC weight increasing)administrator. Here, network namespaces as implemented in OpenStack have been explored along with the mindset that the OpenStack cloud administrator need to have when troubleshooting issues with OpenStack tenant networks. Now, you may want to check out a slideshow on this topic.