Understanding Packet Flows in OpenStack Neutron

April 18, 2014

Version 2.1-209-gbaef0a7

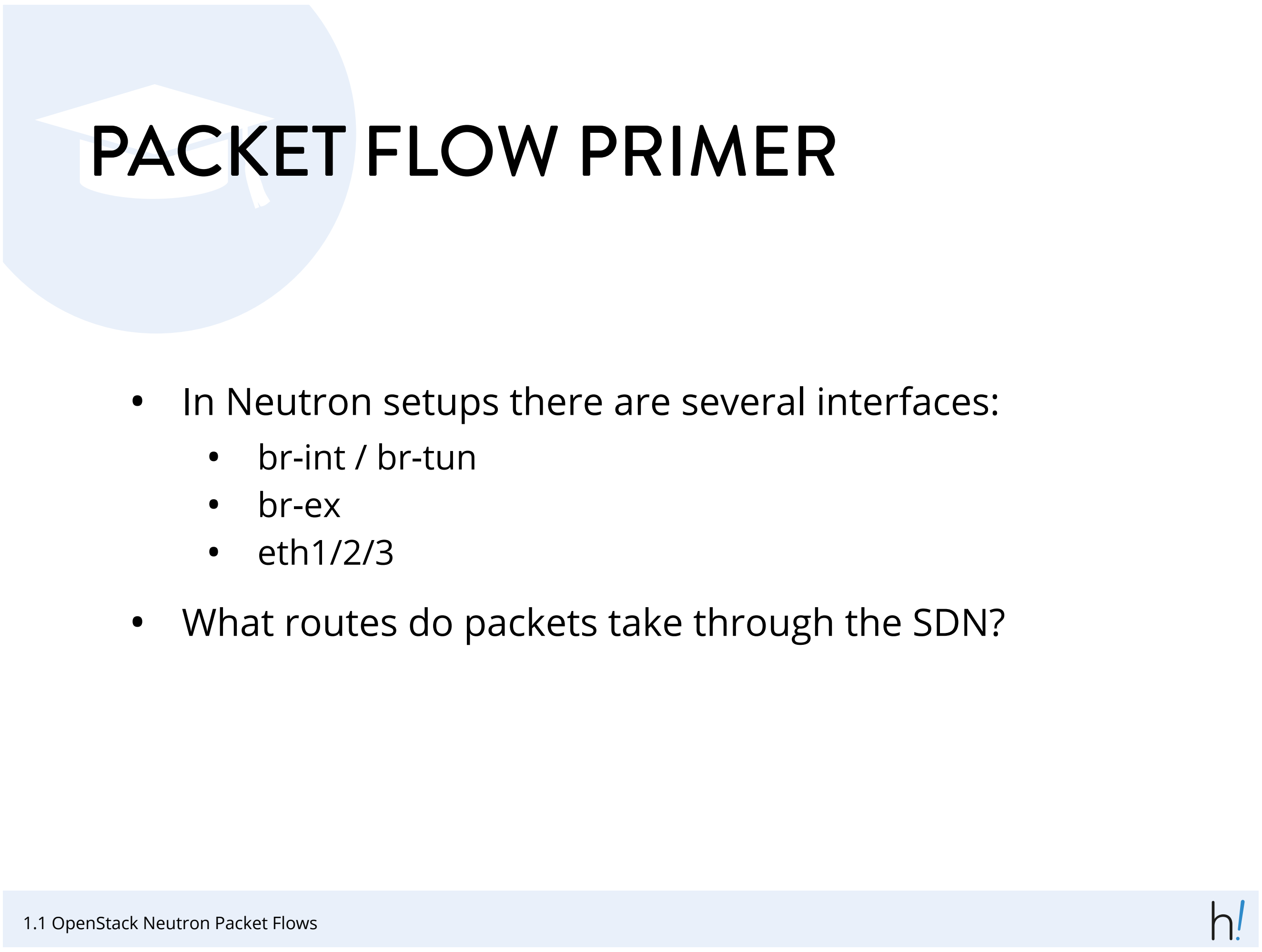
hastexo!

COURSE OUTLINE

1 OpenStack Neutron Packet Flows 1

OpenStack Neutron Packet Flows

1 Understanding Packet Flows in OpenStack Neutron

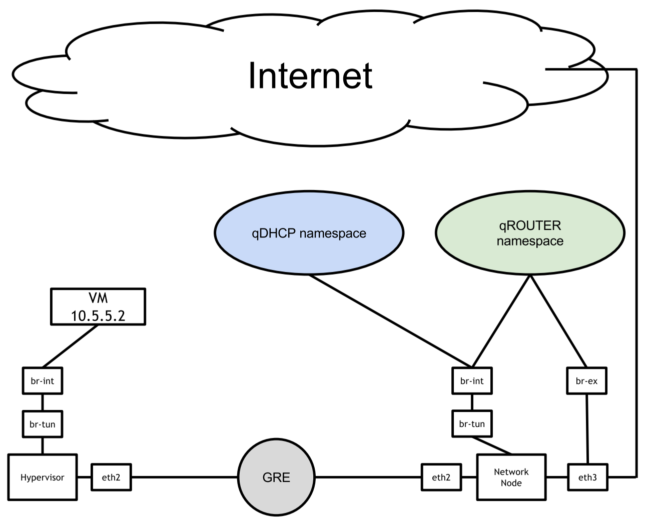


PACKET FLOW PRIMER

- In Neutron setups there are several interfaces:
  - br-int / br-tun
  - br-ex
  - eth1/2/3
- What routes do packets take through the SDN?

A Neutron setup is composed of numerous interfaces, such as br-int, br-tun, br-x, eth1/2/3. For beginners it's usually hard to understand what route packets will take through these devices and hosts. So let's take a closer look.

STANDARD SITUATION



This is the Neutron standard setup. We have a hypervisor node with:

- br-int: This is what the VMs connect to; every VM running on a hypervisor will have its virtual network interface attached to br-int.
- br-tun: This is the tunnel interface that, in GRE mode, adds GRE tunnel headers to the packets.
- eth2: The actual internal physical interface where packets send out via eth2 go to.
- The GRE tunnel: virtual only and would also be present in all hypervisor nodes as well as the network node.

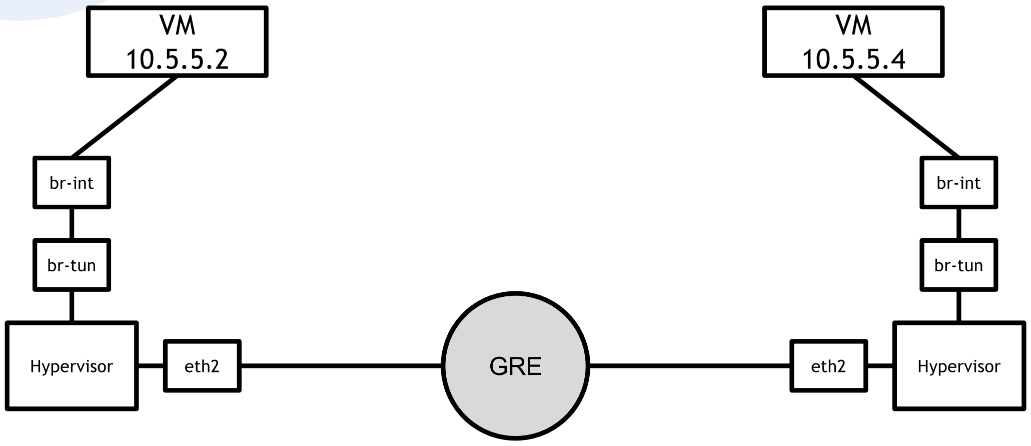
We also have a network node with:

- br-tun/br-int, which serve the same purpose they serve on the hypervisor node
- br-ex physically attached on top of eth3, which is used for communication to the outside

The following network namespaces exist:

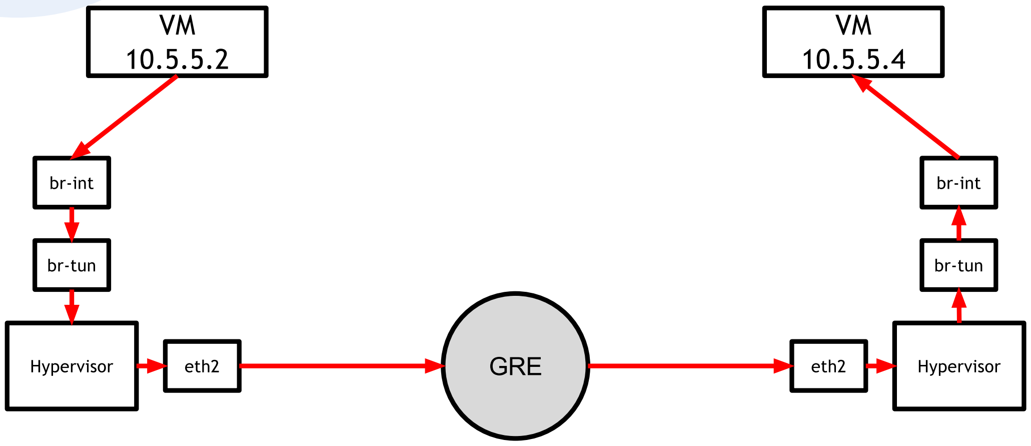
- qDHCP as interface within the GRE tunnels (one for every tenant network)
- qROUTER for connections to the outside (one for every external network)

INTER-VM TRAFFIC



Now, let's take a look at what route packets take. Let's start with an easy example. For now, let's ignore external, internet-bound traffic and focus on traffic happening between VMs. First of all, here is a standard connection scheme between two hypervisor nodes. In our example, the two hypervisor nodes are bob and daisy. The packets will actually take ...

INTER-VM TRAFFIC (CONT.)



this way. They come in on br-int, which adds an OpenFlow ID, then gues to br-tun, which is the tunneling bridge, then leave via eth2. On the other host, it's exactly the same, just vice versa.

OVS OUTPUT: BOB

- `ovs-vsctl show` on `bob` will reveal two virtual switches
  - `br-int`
  - `br-tun`

We can also follow the packet flow based on the actual configuration of OpenFlow and Open vSwitch. Let's have a closer look.

OVS OUTPUT: BOB (CONT.)

```
ovs-vsctl show
[...]
    Bridge br-int
        Port patch-tun
            Interface patch-tun
                options: {peer=patch-int}
        Port "tape4dbc657-c4"
            tag: 1
            Interface "tape4dbc657-c4"
        Port br-int
            Interface br-int
```

Here, the tap interface reveals that packets coming in from that interface will get an OpenFlow ID of 1.

And thanks to `patch-tun`, we know that packets in this switch will automatically be send to the `patch-int` interface as well, which happens to be part of the `br-tun` bridge.

Last but not least, `br-int` represents the actual bridge interface in the network stack.

OVS OUTPUT: BOB (CONT.)

```
ovs-vsctl show
[...]
Bridge br-tun
  Port br-tun
    Interface br-tun
  Port patch-int
    Interface patch-int
        options: {peer=patch-tun}
  Port "gre-1"
    Interface "gre-1"
       options: {in_key=flow, local_ip="192.168.133.112", \
               out_key=flow, remote_ip="192.168.133.114"}
```

Inside of `br-tun`, we see the according interface for `patch-int`. We also see `br-tun`, which represents the actual bridge interface in the network stack.

And then there is `gre-1`, which is an actual GRE tunnel, in this case pointing to `daisy`, an imaginary second hypervisor node. Thanks to this information, we know that packets coming from the VM will leave bob via the `gre-1` interface in an encapsulated manner.

OVS OUTPUT: DAISY

- Now, let's take a look at daisy, our imaginary second hyprvisor.
- The output of `ovs-ofctl dump-flows br-tun` will help us understand what happens with packets coming in on the `br-tun` interface on `daisy`

If you want to know what happens with packets coming in on a specific interface, follow OpenFlow flows using `ovs-ofctl dump-flows` on the virtual switch, in this case `br-tun`

OVS OUTPUT: DAISY (CONT.)

```
ovs-ofctl dump-flows br-tun
[...]
 cookie=0x0, duration=27546.491s, table=2, \
     n_packets=1537, n_bytes=359538, idle_age=25, \
     priority=1,tun_id=0x1 \
     actions=mod_vlan_vid:1,resubmit(,10)
```

This tells us that in the OpenFlow flows of `br-tun` on `daisy`, packets with the OpenFlow ID 1 will be redirected from the OpenFlow table 2 to the OpenFlow table 10

OVS OUTPUT: DAISY (CONT.)

```
ovs-ofctl dump-flows br-tun
[...]
 cookie=0x0, duration=27547.332s, table=10, \
     n_packets=1537, n_bytes=359538, \
     idle_age=25, priority=1 \
     actions=[...],output:1
```

Inside of OpenFlow Table 10, we see the important part at the end of the `actions` parameter: The output of the packet will happen on the virtual switch port 1

OVS OUTPUT: DAISY (CONT.)

- Figuring out which network interface represents a specific port of a virtual switch:
  - `ovs-ofctl show br-tun`

```
1(patch-int): addr:76:7c:cb:84:54:f3
[...]
2(gre-2): addr:16:14:65:eb:26:8c
[...]
LOCAL(br-tun): addr:1e:5e:83:65:02:4e
[...]
```

So port 1 is `patch-int`. The numbers in the brackets indicate the port interfaces.

OVS OUTPUT: DAISY (CONT.)

- Next, we need to figure out what patch-int actually is. Again, the output of `ovs-vsctl show`, this time on daisy, comes in handy.

```
Bridge br-tun
    [...]
    Port patch-int
        Interface patch-int
            options: {peer=patch-tun}
    [...]
```

- So in fact, packets coming in on `patch-tun` will be forwarded to the `patch-int` port of the virtual switch

Actually, the design of `br-tun` and `br-int` on `daisy` is identical to the one on `bob`

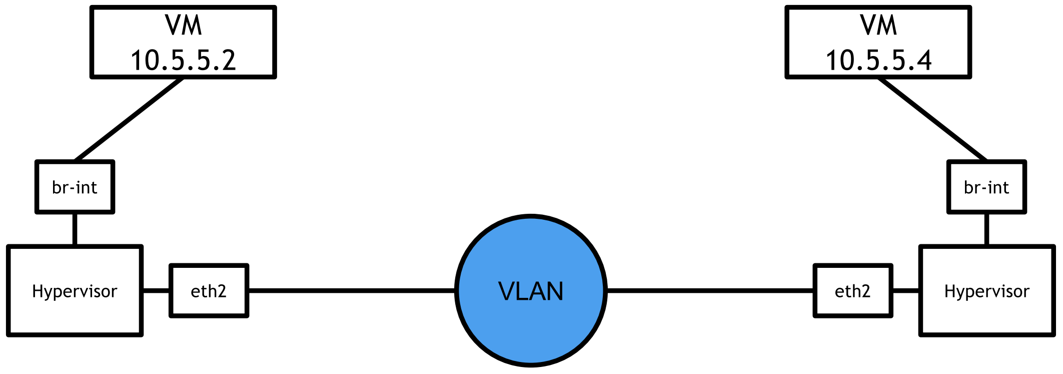
OVS OUTPUT: DAISY (CONT.)

- Similar to what we saw on `bob`, the situation is on `daisy`:

```
ovs-vsctl show
[...]
    Bridge br-int
        Port "tap6dcc748c-17"
            tag: 1
        [...]
        Port patch-tun
                options: {peer=patch-int}
```

So in fact, packets coming in via `patch-tun` to `br-int` will, if they have an OpenFlow ID of 1, be forwarded to the port `tap6dcc748c-17`. For every other VM, a similar entry would exist.

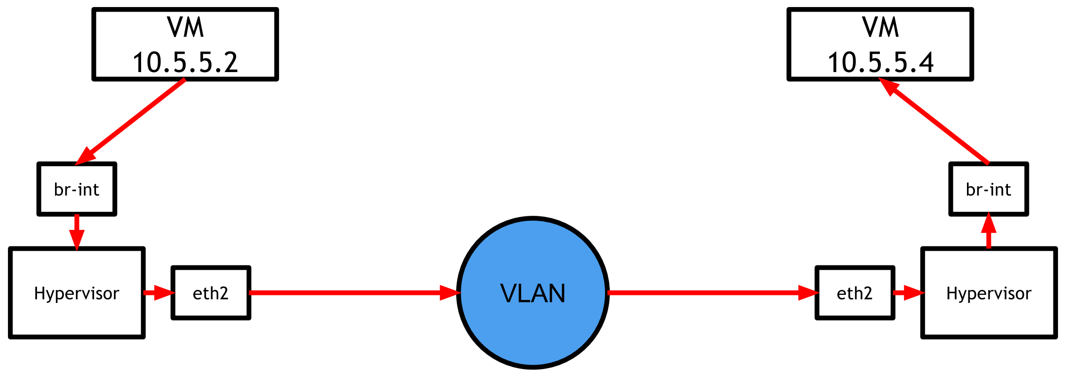
INTER-VM: VLAN



1.1 OpenStack Neutron Packet Flows

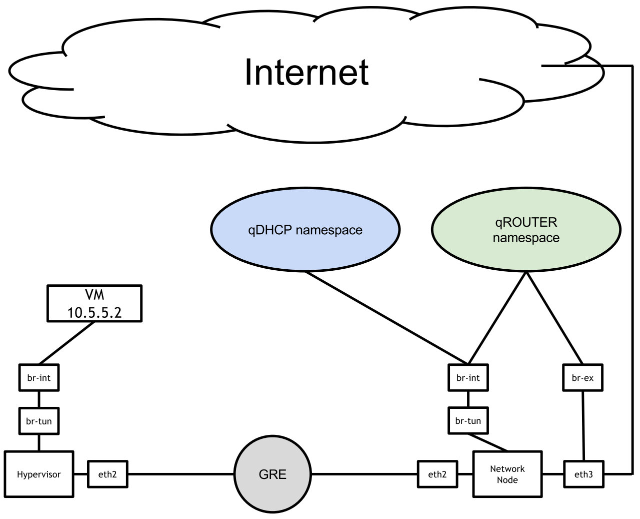
Oh and while we are at it. It's important to understand that in this case, GRE really is only a transport and does not influence the actual SDN functionality. If you wanted to use VLANs instead of GRE tunneling rather, then that wouldn't be the least bit of a problem with OpenFlow. You would not have a tunneling bridge `br-tun`, instead, `br-int` would have VLAN-tagged interfaces. The bridging ethernet interface would be directly connected to a switch; this link could either be a trunk link or a link with all VLANs enabled (which is tedious and doesn't scale, but still some companies do it that way).

INTER-VM: VLAN (CONT.)



The outcome, however, is identical to the VLAN scenario. VMs can talk to each other securely.

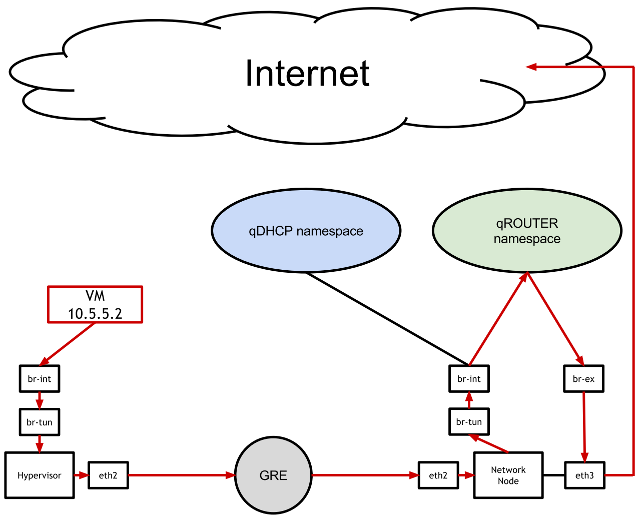
VMS ACCESSING THE INTERNET



1.1 OpenStack Neutron Packet Flows

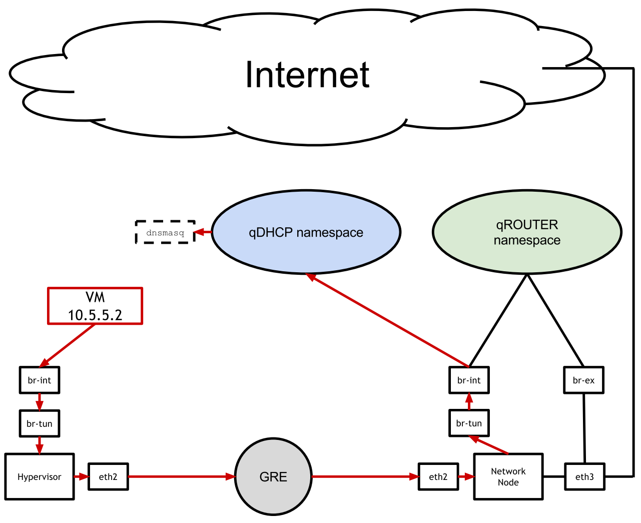
Let's add the internet back to this. How does internet traffic originating in VMs actually work?

VMS ACCESSING THE INTERNET (CONT.)



1. The packet leaves the VM and hits the hypervisor node on br-int
2. From there, it is forwarded to br-tun, which adds the GRE encapsulation to it
3. It leaves the hypervisor on eth2 into the GRE network
4. It reaches eth2 on the network node
5. It gets sent through br-tun, which removes the GRE Headers
6. It is then forwarded to br-int (br-int on the network node is connected to the qROUTER namespace -- that is why we defined our external network as router for the admin tenant network earlier)
7. From br-int, the packet hits the external bridge (br-ex) and finally leaves the network node into the internet

VMS SENDING DHCP REQUESTS

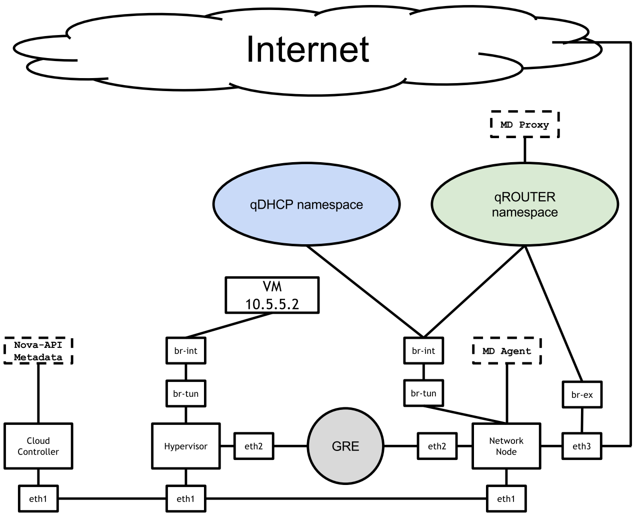


1.1 OpenStack Neutron Packet Flows

As another example, let's see what happens when VMs send DHCP requests:

1. The packet leaves the VM and hits the hypervisor node on br-int
2. From there, it is forwarded to br-tun, which adds the GRE encapsulation to it
3. It leaves the hypervisor on eth2 into the GRE network
4. It reaches eth2 on the network node
5. It gets send through br-tun, which removes the GRE Headers
6. It is then forwarded to br-int (br-int on the network node is also connected to the qDHCP namespace -- the dnsmasq processes are running within it)
7. The qDHCP request is then answered accordingly

VMS LOOKING FOR METADATA



1.1 OpenStack Neutron Packet Flows

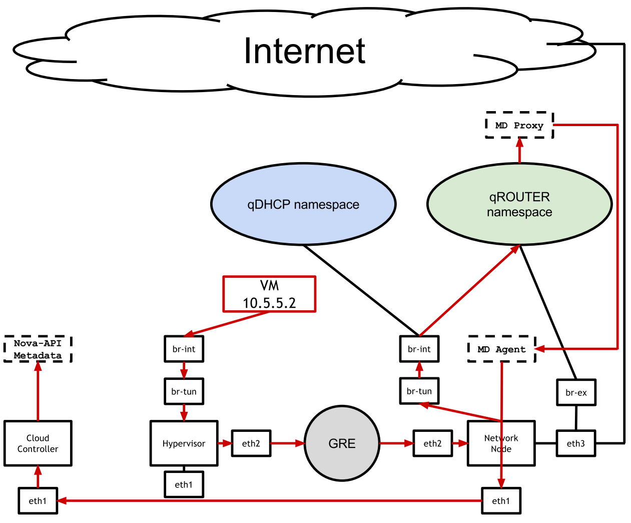
To get a better understanding of the process, first, we need to add the cloud controller to our scheme here. We also need to add the so-called management network, which all nodes are physically connected to via the eth1 physical interface.

Also, we have the Nova-API metadata service running on our cloud controller. This is the source of metadata, the service that VMs eventually need to connect to to get it.

And then, we have the metadata-agent and the metadata proxy running on the network node.

So, how do packets flow here?

VMS LOOKING FOR META-DATA (CONT.)



Let's start with the actual request coming from the VM. It makes its usual way and will eventually make it to the qROUTER namespace. The destination is IP 169.254.169.254, and thus, the packet leaves the VM via its default route, which is an IP inside the qROUTER namespace.

Within the qROUTER namespace, the package will be forwarded via DNAT from port 80 to the qrouter's port 9697, which is where the metadata proxy is running. The metadata proxy will relay the packets to the metadata agent running on the network node.

The latter part happens outside of any TCP/IP network connectivity: the metadata agent has a UNIX socket open in /var/lib/neutron/metadata_proxy which the metadata proxy pipes packets into.

COPYRIGHT NOTICE