



Red Hat OpenStack Platform 14

Use Skydive for OpenStack network analysis

Use Skydive to view, troubleshoot, and audit your OpenStack network

Red Hat OpenStack Platform 14 Use Skydive for OpenStack network analysis

Use Skydive to view, troubleshoot, and audit your OpenStack network

OpenStack Team
rhos-docs@redhat.com

Legal Notice

Copyright © 2019 Red Hat, Inc.

The text of and illustrations in this document are licensed by Red Hat under a Creative Commons Attribution–Share Alike 3.0 Unported license ("CC-BY-SA"). An explanation of CC-BY-SA is available at

<http://creativecommons.org/licenses/by-sa/3.0/>

. In accordance with CC-BY-SA, if you distribute this document or an adaptation of it, you must provide the URL for the original version.

Red Hat, as the licensor of this document, waives the right to enforce, and agrees not to assert, Section 4d of CC-BY-SA to the fullest extent permitted by applicable law.

Red Hat, Red Hat Enterprise Linux, the Shadowman logo, JBoss, OpenShift, Fedora, the Infinity logo, and RHCE are trademarks of Red Hat, Inc., registered in the United States and other countries.

Linux ® is the registered trademark of Linus Torvalds in the United States and other countries.

Java ® is a registered trademark of Oracle and/or its affiliates.

XFS ® is a trademark of Silicon Graphics International Corp. or its subsidiaries in the United States and/or other countries.

MySQL ® is a registered trademark of MySQL AB in the United States, the European Union and other countries.

Node.js ® is an official trademark of Joyent. Red Hat Software Collections is not formally related to or endorsed by the official Joyent Node.js open source or commercial project.

The OpenStack ® Word Mark and OpenStack logo are either registered trademarks/service marks or trademarks/service marks of the OpenStack Foundation, in the United States and other countries and are used with the OpenStack Foundation's permission. We are not affiliated with, endorsed or sponsored by the OpenStack Foundation, or the OpenStack community.

All other trademarks are the property of their respective owners.

Abstract

Use Skydive to view, troubleshoot, and audit your OpenStack network.

Table of Contents

CHAPTER 1. SKYDIVE OVERVIEW	3
1.1. USE CASES FOR OPENSTACK DEPLOYMENTS	3
1.2. SKYDIVE ARCHITECTURE	4
1.3. DATA SOURCES	4
CHAPTER 2. INSTALL SKYDIVE	5
CHAPTER 3. USING THE SKYDIVE UI	6
CHAPTER 4. USING THE SKYDIVE COMMAND LINE	9
4.1. CHECK SKYDIVE STATUS	9
CHAPTER 5. CAPTURE TRAFFIC STATISTICS	12

CHAPTER 1. SKYDIVE OVERVIEW



IMPORTANT

Skydive is included with this release as a Technology Preview. For more information on the support scope for features marked as technology previews, see <https://access.redhat.com/solutions/21101>.

A typical software-defined networking (SDN) deployment can grow to significant complexity, as the layers of abstraction and virtualization make network analysis and troubleshooting seem more difficult. This complexity can be apparent in a cloud product such as OpenStack, where it can be a challenge to track the flow of traffic, or distinguish between the various types of bridged and tunneled networks that connect instances.

To help with these challenges, Skydive is a lightweight distributed service that allows you view and debug your SDN deployment, allowing for easier network analysis and troubleshooting. Skydive includes a web UI that displays a live view of your network topology, dependencies, and flows. The Skydive CLI allows you to generate reports and perform configuration audits. Skydive also includes a REST API; for more information, see <http://skydive.network/documentation/api>.



NOTE

Skydive is intended for use by cloud administrators only, and is not intended as a service for end-users or individual tenants.

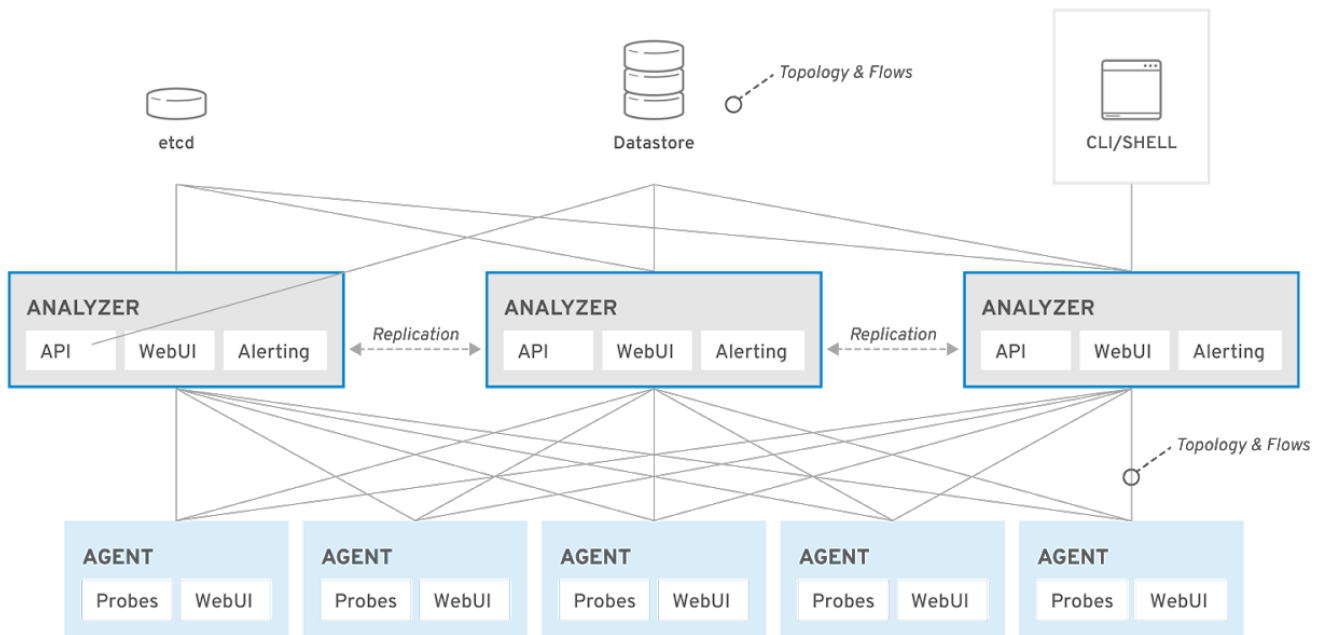
1.1. USE CASES FOR OPENSTACK DEPLOYMENTS

Use the data presented by Skydive for a number of use cases:

- Troubleshooting
 - Packet loss
 - Performance issues
 - Review ingress and egress flows
- Validate services and configuration
 - Check that your deployment is working correctly.
 - Review whether the expected MTU settings are correctly applied.
- Network Monitoring
 - Generate traffic reports for a particular tenant.
 - Capture network traffic between virtual interfaces.
 - Alarming - for example: when an interface drops, or if a flow reaches a certain percentage of utilization.
- Capacity Planning
 - Identify low-latency nodes to host certain services.

1.2. SKYDIVE ARCHITECTURE

Skydive is a distributed service that relies on agents installed on each node. As demonstrated in the following diagram, these agents send data back to the analyzer services. The analyzer service reviews the data, generates reports, and present the results using an API. These reports can then be viewed using the web UI or through the command line reporting tool.



Skydive is installed using Red Hat OpenStack Platform director, and is available as two composable services. For more information, see [Chapter 2, Install Skydive](#).

1.3. DATA SOURCES

To generate its reports, Skydive analyzes the following components:

- **OVSDB** - Open vSwitch (OVS) components. This information includes the OVSDB, bridges, ports, and interfaces.
- **Netlink** - A node's network objects. This information includes the host's interfaces, network namespaces, bridges, MTU, among others.
- **neutron** - The OpenStack networking service.
- Container services.
- Custom objects - You can add your own static network components, such as a top-of-rack (ToR) switch.

CHAPTER 2. INSTALL SKYDIVE



IMPORTANT

Skydive is included with this release as a Technology Preview. For more information on the support scope for features marked as technology previews, see <https://access.redhat.com/solutions/21101>.

Skydive's two components, the analyzer and the agent, are available as composable services:

```
OS::TripleO::Services::SkydiveAgent
OS::TripleO::Services::SkydiveAnalyzer
```

Install Skydive by adding **skydive-environment.yaml** to your **openstack overcloud deploy** command. The director installs **SkydiveAgent** on each Controller and Compute node by default. These agents report the network analytics back to **SkydiveAnalyzer**, running on the Controller node.

CHAPTER 3. USING THE SKYDIVE UI

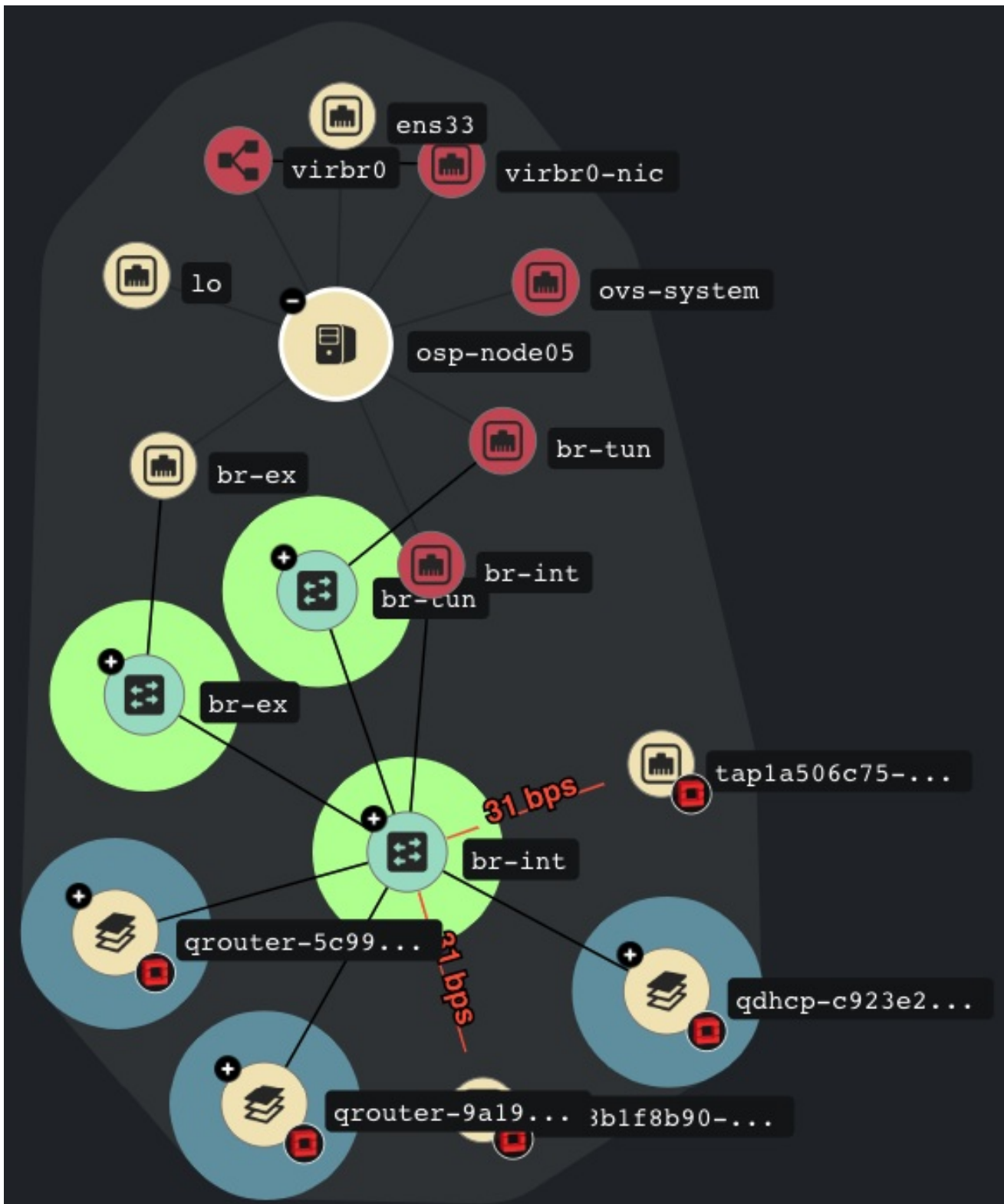
Use the Skydive UI to perform a number of tasks, such as viewing your network topology. By default, you access the UI through a web browser on port **8082**. For example:

`https://192.168.123.141:8082`.

The default view shows all your physical nodes. Click the **Expand** button to see the full view for a node:



The expanded node displays the virtual networking topology:



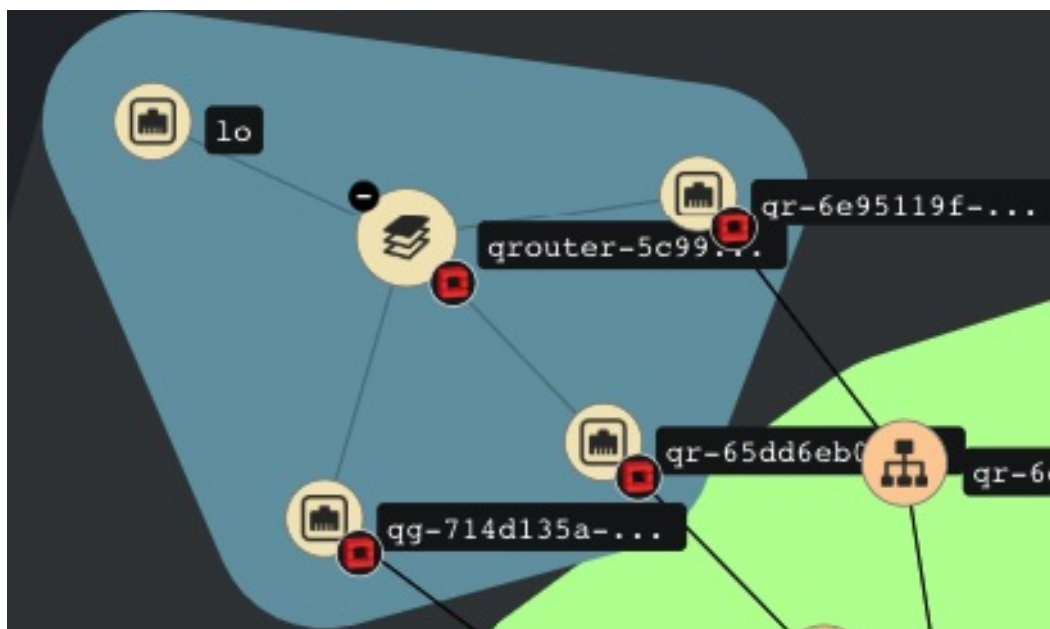
In this example, the physical node **osp-node05** contains various connection types; these are presented in different colors to distinguish their functions:

- **Beige:** The physical NICs and the OVS interface `br-ex`
- **Blue:** The network namespaces created for the OpenStack projects.
- **Red:** Indicates that the network interface is down.

To see additional details, click the same **Expand** button again:



This view reveals further details of the network topology, such as the instance interfaces within a Project's namespace:



CHAPTER 4. USING THE SKYDIVE COMMAND LINE

The Skydive CLI allows you to perform the same functions as the GUI. For example:

- Review your topology
- Capture traffic
- Perform packet injection

4.1. CHECK SKYDIVE STATUS

You can use `skydive client status` to check the state of your Skydive services. For example:

```
$ skydive client status
{
  "Agents": {
    "node01.lab.local": {
      "ServiceType": "agent",
      "ClientProtocol": "protobuf",
      "Addr": "10.0.0.20",
      "Port": 49454,
      "IsConnected": true,
      "ConnectTime": "2018-11-16T09:19:12.652607Z",
      "RemoteHost": "node01.lab.local"
    },
    "node02.lab.local": {
      "ServiceType": "agent",
      "ClientProtocol": "protobuf",
      "Addr": "10.0.0.9",
      "Port": 43916,
      "IsConnected": true,
      "ConnectTime": "2018-11-16T09:19:12.770824379Z",
      "RemoteHost": "node02.lab.local"
    },
    "node1": {
      "ServiceType": "agent",
      "ClientProtocol": "protobuf",
      "Addr": "192.168.42.132",
      "Port": 54582,
      "IsConnected": true,
      "ConnectTime": "2018-11-16T09:19:12.803125487Z",
      "RemoteHost": "node1"
    }
  },
  "Peers": {
    "Incomers": {},
    "Outgoers": {}
  },
  "Publishers": {},
  "Subscribers": {},
  "Alerts": {
    "IsMaster": true
  },
  "Captures": {
```

```
    "IsMaster": true
  },
  "Probes": [
    "fabric",
    "peering"
  ]
}
```

Skydive includes a command line reporting tool that uses the Gremlin syntax. Gremlin searches the Skydive database to help review your system configuration. For example, this query returns a list of IP addresses that do not use a MTU size of exactly **1500**:

```
$ /opt/stack/go/bin/skydive client query "G.V().Has('MTU',
NE(1500)).Values('IPV4')"
```

```
[
  [
    "127.0.0.1/8"
  ],
  [
    "127.0.0.1/8"
  ],
  [
    "10.0.0.9/24"
  ],
  [
    "10.233.64.1/24"
  ],
  [
    "127.0.0.1/8"
  ],
  [
    "127.0.0.1/8"
  ],
  [
    "10.233.64.4/24"
  ],
  [
    "10.0.0.3/26"
  ],
  [
    "127.0.0.1/8"
  ],
  [
    "10.0.0.1/26"
  ],
  [
    "127.0.0.1/8"
  ],
  [
    "127.0.0.1/8"
  ],
  [
    "10.233.64.0/32"
  ],
  [
    "10.0.0.2/26"
  ]
]
```

```

],
[
  "10.0.0.20/24",
  "192.168.0.1/24"
],
[
  "10.233.64.2/24"
]
]

```

To see which IP addresses use a MTU of exactly **1500**:

```

$ /opt/stack/go/bin/skydive client query
"G.V().Has('MTU',1500).Values('IPV4')"
[
  [
    "192.168.122.1/24"
  ],
  [
    "192.168.42.129/25"
  ],
  [
    "172.17.0.1/16"
  ],
  [
    "172.17.0.1/16"
  ],
  [
    "192.168.42.132/25"
  ],
  [
    "192.168.122.1/24"
  ]
]

```

For more information on:

- Gremlin queries, see <http://skydive.network/documentation/api-gremlin>.
- Skydive CLI, see <http://skydive.network/documentation/cli>.
- Skydive Rest API, see <http://skydive.network/documentation/api-rest>.

CHAPTER 5. CAPTURE TRAFFIC STATISTICS

Use Skydive to capture the network statistics of an instance's interface. This is useful for troubleshooting and diagnostics, or determining whether the network works as expected. To do this, first create a capture between the two interfaces, then use the **Flows** tool to query the captured traffic.

This example shows an ICMP ping between **instance-1** and **instance-2**. The intention is to observe these ICMP packets in a monitoring session:

1. Review the IP addresses assigned to the instances:

```
$ openstack server list
+-----+-----+-----+-----+-----+-----+
| ID                | Name          | Status |
+-----+-----+-----+-----+-----+-----+
| 24a20f49-a23a-4f30-b1c3-d67a8277e42f | instance-2 | ACTIVE |
| web=192.168.201.19 |             | m1.nano |
+-----+-----+-----+-----+-----+-----+
| eea0a3cc-9338-4f01-bc72-df3252a5c689 | instance-1 | ACTIVE |
| private=192.168.200.3 |            | m1.nano |
+-----+-----+-----+-----+-----+-----+
```

2. In the Skydive UI, click **Capture** in the right-hand pane and click **Create**.
3. Click **Gremlin Expression** to create a query to capture traffic based on custom criteria. For example, you could enter one of the following into the **Query** field:

- If you know vm-id of the instance: `G.V().Has('ExtID.vm-id', '24a20f49-a23a-4f30-b1c3-d67a8277e42f')`
- If you know the IP address of the instance: `G.V().Has('Neutron.IPV4', IPRange('10.0.0.3/32'))`
- If you know the IP address of the interface: `G.V().Has('IPV4', '10.0.0.20/24')`
- If you know the IP address but not the mask: `G.V().Has('IPV4', IPRANGE('10.0.0.20/32'))`



NOTE

Edit the IP address or instance ID to suit your deployment.

4. Click **Start** to begin the capture.
For example, adding the ID of **instance-2** to the query: `G.V().Has('ExtID.vm-id', '24a20f49-a23a-4f30-b1c3-d67a8277e42f')`, the **Flows** table shows all network traffic for that instance:

Captures Generator Flows Alerts Workflows Topology rules

540899c9-0f4d-4c87-4e28-530bc8d5108b

Query `G.V().Has('ExtID.vm-id', '24a20f49-a23a-4f30-b1c3-d67a8277e42f')`

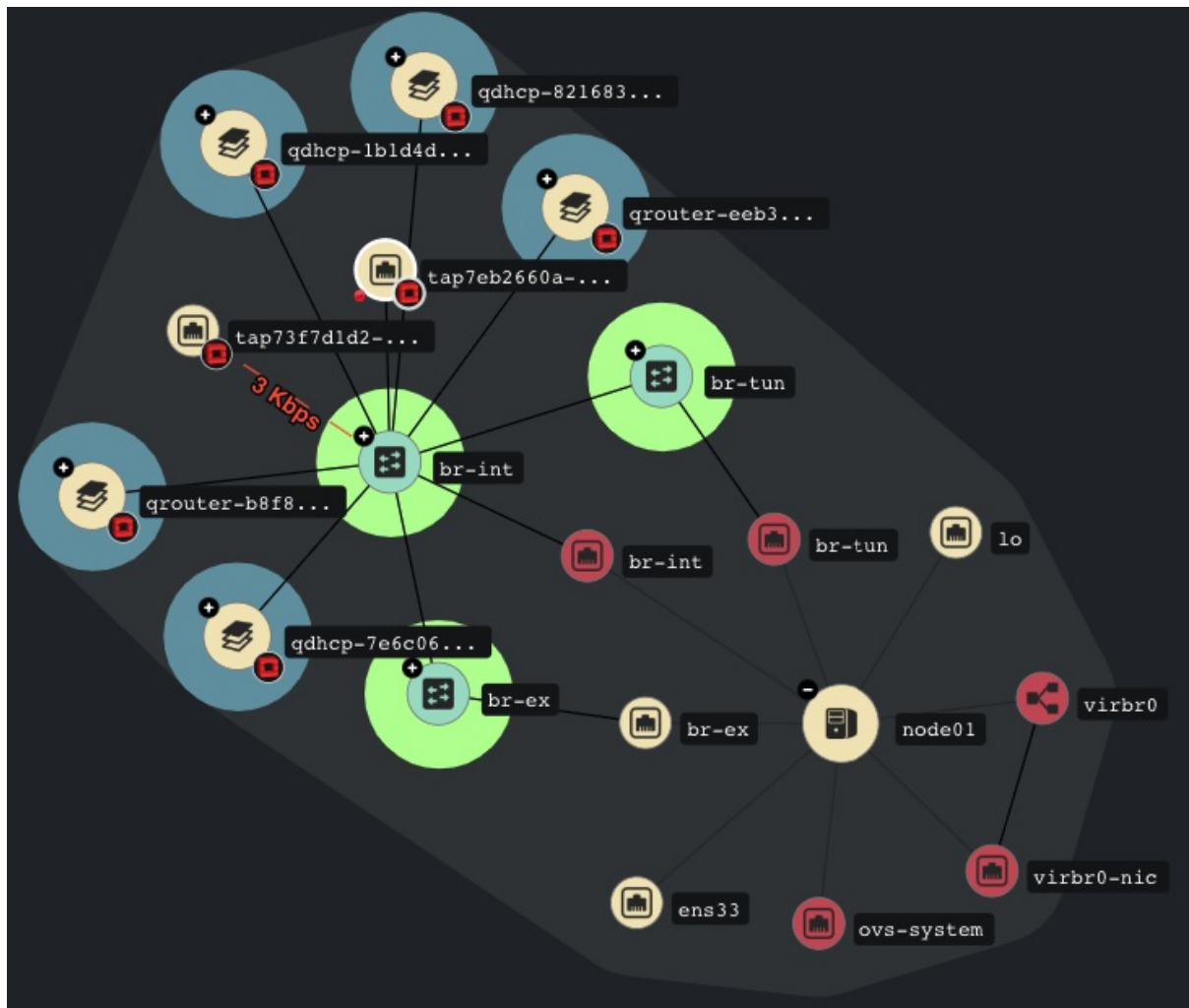
Layer mode L2

Flows

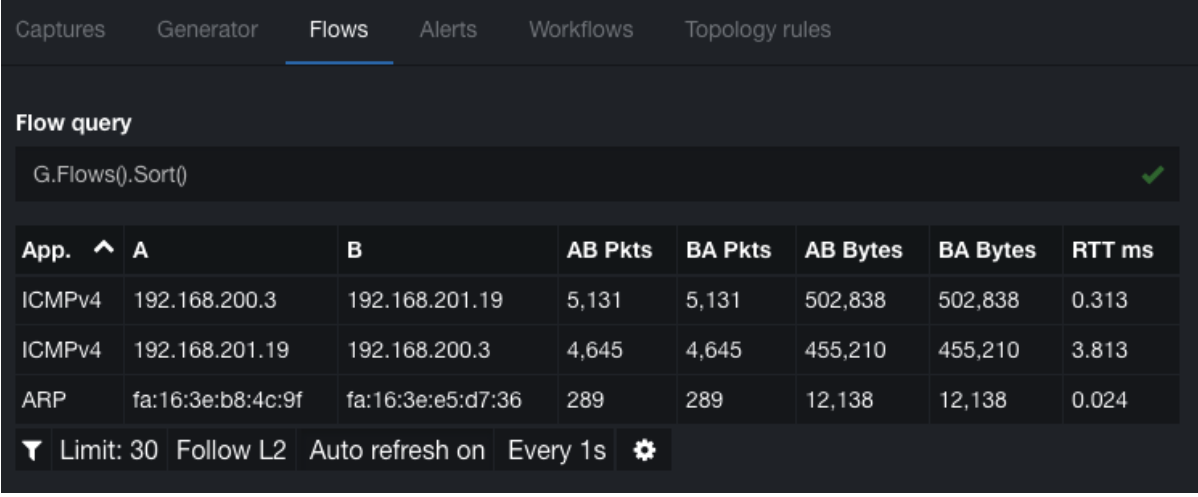
App. ^	A	B	AB Pkts	BA Pkts	AB Bytes	BA Bytes	RTT ms
ICMPv4	192.168.200.3	192.168.201.19	28	28	2,744	2,744	0.313
ARP	fa:16:3e:b8:4c:9f	fa:16:3e:e5:d7:36	1	1	42	42	0.024

Limit: 30 Follow L2 Auto refresh on Every 1s

Skydive automatically selects the topology in the diagram. The instance interfaces are indicated by the OpenStack logo:



- To view the packet statistics, click on **Flows** at the bottom of the right-hand pane. Consider enabling **Auto refresh** for an updated view. This example shows the network traffic moving between **instance-1** and **instance-2**:



The screenshot shows the 'Flows' tab in the Skydive interface. At the top, there are navigation tabs: Captures, Generator, Flows (selected), Alerts, Workflows, and Topology rules. Below the tabs is a 'Flow query' input field containing the text 'G.Flows().Sort()' with a green checkmark to its right. Underneath the query field is a table with the following data:

App. ^	A	B	AB Pkts	BA Pkts	AB Bytes	BA Bytes	RTT ms
ICMPv4	192.168.200.3	192.168.201.19	5,131	5,131	502,838	502,838	0.313
ICMPv4	192.168.201.19	192.168.200.3	4,645	4,645	455,210	455,210	3.813
ARP	fa:16:3e:b8:4c:9f	fa:16:3e:e5:d7:36	289	289	12,138	12,138	0.024

At the bottom of the table, there is a control bar with a dropdown arrow, 'Limit: 30', 'Follow L2', 'Auto refresh on', 'Every 1s', and a gear icon for settings.

6. By default, the *Flows* UI uses the Gremlin query `G.Flows().Sort()`, which returns all the captured traffic. Enter your own query into the field to refine this view. For example:

- Return only ICMPv4 traffic: `G.Flows().Has('Application', 'ICMPv4').Limit(10)`