Red Hat OpenStack Platform 13

Networking with Open Virtual Network

OpenStack Networking with OVN

Last Updated: 2021-10-27

Red Hat OpenStack Platform 13 Networking with Open Virtual Network

OpenStack Networking with OVN

OpenStack Team
rhos-docs@redhat.com

Legal Notice

Abstract

A Cookbook for using OVN for OpenStack Networking Tasks.

Table of Contents

MAKING OPEN SOURCE MORE INCLUSIVE 3

CHAPTER 1. OPEN VIRTUAL NETWORK (OVN) 4
1.1. LIST OF COMPONENTS IN THE RHOSP OVN ARCHITECTURE 4

CHAPTER 2. PLANNING YOUR OVN DEPLOYMENT 6
2.1. THE OVN-CONTROLLER SERVICE ON COMPUTE NODES 6
2.2. THE OVN COMPOSABLE SERVICE 6
2.3. DEPLOYING A CUSTOM ROLE WITH ML2/OVN 7
2.4. HIGH AVAILABILITY WITH PACEMAKER AND DVR 8
2.5. LAYER 3 HIGH AVAILABILITY WITH OVN 8

CHAPTER 3. DEPLOYING OVN WITH DIRECTOR 10
3.1. DEPLOYING ML2/OVN WITH DVR 10
3.2. DEPLOYING THE OVN METADATA AGENT ON COMPUTE NODES 11
3.2.1. Troubleshooting Metadata issues 11
3.3. DEPLOYING INTERNAL DNS WITH OVN 11

CHAPTER 4. MONITORING OVN 13
4.1. CREATING ALIASES FOR OVN TROUBLESHOOTING COMMANDS 13
4.2. MONITORING OVN LOGICAL FLOWS 14
4.3. MONITORING OPENFLOWS 16

MAKING OPEN SOURCE MORE INCLUSIVE

Red Hat is committed to replacing problematic language in our code, documentation, and web properties. We are beginning with these four terms: master, slave, blacklist, and whitelist. Because of the enormity of this endeavor, these changes will be implemented gradually over several upcoming releases. For more details, see our CTO Chris Wright's message.

CHAPTER 1. OPEN VIRTUAL NETWORK (OVN)

Open Virtual Network (OVN) is an Open vSwitch-based software-defined networking (SDN) solution for supplying network services to instances. OVN provides platform-neutral support for the full OpenStack Networking API. With OVN, you can programatically connect groups of guest instances into private L2 and L3 networks. OVN uses a standard approach to virtual networking that is capable of extending to other Red Hat platforms and solutions.

This release of the Red Hat OpenStack Platform (RHOSP) does not provide a supported migration from the ML2/OVS mechanism driver to the ML2/OVN mechanism driver. This RHOSP release does not support the OpenStack community migration strategy. Migration support is planned for a future RHOSP release.

NOTE

The minimum OVS version required is OVS 2.9.

This section describes the steps required to deploy OVN using director.

NOTE

OVN is supported only in a a RHOSP high availability (HA) environment with at least three controller nodes with distributed virtual routing (DVR).

1.1. LIST OF COMPONENTS IN THE RHOSP OVN ARCHITECTURE

The RHOSP OVN architecture replaces the OVS Modular Layer 2 (ML2) mechanism driver with the OVN ML2 mechanism driver to support the Networking API. OVN provides networking services for the Red Hat OpenStack platform.

The OVN architecture consists of the following components and services:

ML2 plugin with OVN mechanism driver

The ML2 plug-in translates the OpenStack-specific networking configuration into the platform-neutral OVN logical networking configuration. It typically runs on the Controller node.

OVN Northbound (NB) database (ovn-nb)

This database stores the logical OVN networking configuration from the OVN ML2 plugin. It typically runs on the Controller node and listens on TCP port **6641**.

OVN Northbound service (ovn-northd)

This service converts the logical networking configuration from the OVN NB database to the logical data path flows and populates these on the OVN Southbound database. It typically runs on the Controller node.

OVN Southbound (SB) database (ovn-sb)

This database stores the converted logical data path flows. It typically runs on the Controller node and listens on TCP port **6642**.

OVN controller (ovn-controller)

This controller connects to the OVN SB database and acts as the open vSwitch controller to control and monitor network traffic. It runs on all Compute and gateway nodes where **OS::Tripleo::Services::OVNController** is defined.

OVN metadata agent (ovn-metadata-agent)

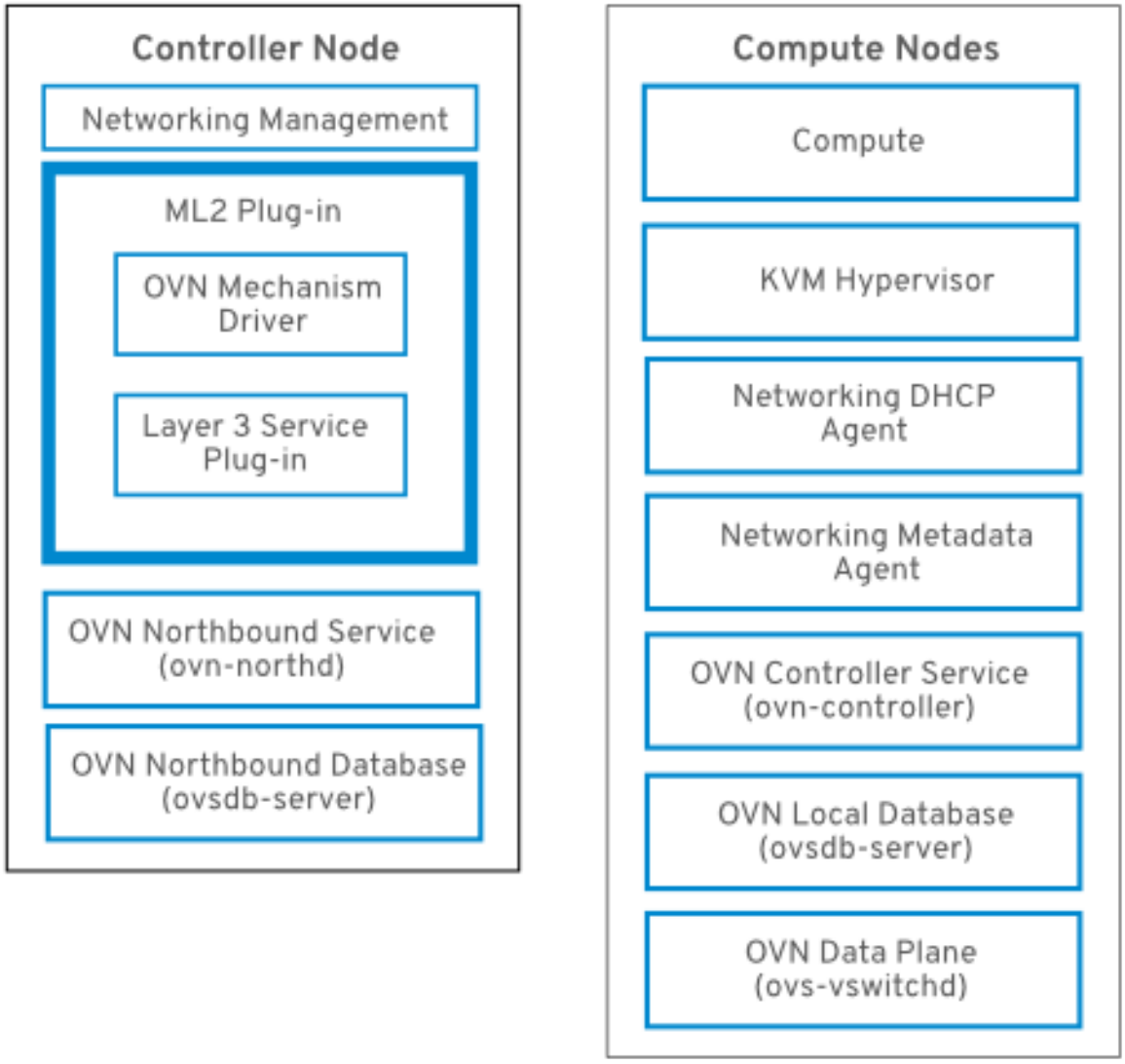
This agent creates the **haproxy** instances for managing the OVS interfaces, network namespaces and HAProxy processes used to proxy metadata API requests. The agent runs on all Compute and gateway nodes where **OS::TripleO::Services::OVNMetadataAgent** is defined.

OVS database server (OVSDB)

Hosts the OVN Northbound and Southbound databases. Also interacts with **ovs-vswitchd** to host the OVS database **conf.db**.

NOTE

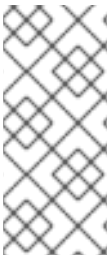
The schema file for the NB database is located in **/usr/share/ovn/ovn-nb.ovsschema**, and the SB database schema file is in **/usr/share/ovn/ovn-sb.ovsschema**.



CHAPTER 2. PLANNING YOUR OVN DEPLOYMENT

Deploy OVN only in high-availability RHOSP high availability (HA) environments with at least three controller nodes. Deploy OVN with distributed virtual routing (DVR) enabled.

DVR is enabled by default in new ML2/OVN deployments and disabled by default in new ML2/OVS deployments. The **neutron-ovn-dvr-ha.yaml** environment file configures the required DVR-specific parameters for deployments using OVN in an HA environment.



NOTE

To use OVN, your director deployment must use Generic Network Virtualization Encapsulation (Geneve), and not VXLAN. Geneve allows OVN to identify the network using the 24-bit Virtual Network Identifier (VNI) field and an additional 32-bit Type Length Value (TLV) to specify both the source and destination logical ports. You should account for this larger protocol header when you determine your MTU setting.

2.1. THE OVN-CONTROLLER SERVICE ON COMPUTE NODES

The **ovn-controller** service runs on each Compute node and connects to the OVN southbound (SB) database server to retrieve the logical flows. The **ovn-controller** translates these logical flows into physical OpenFlow flows and adds the flows to the OVS bridge (**br-int**). To communicate with **ovs-vswitchd** and install the OpenFlow flows, the **ovn-controller** connects to the local **ovsdb-server** (which hosts **conf.db**) using the UNIX socket path that was passed when **ovn-controller** was started (for example **unix:/var/run/openvswitch/db.sock**).

The **ovn-controller** service expects certain key-value pairs in the **external_ids** column of the **Open_vSwitch** table; **puppet-ovn** uses **puppet-vswitch** to populate these fields. The following example shows the key-value pairs that **puppet-vswitch** configures in the **external_ids** column:

```
hostname=<HOST NAME>
ovn-encap-ip=<IP OF THE NODE>
ovn-encap-type=geneve
ovn-remote=tcp:OVN_DBS_VIP:6642
```

2.2. THE OVN COMPOSABLE SERVICE

Red Hat OpenStack Platform usually consists of nodes in pre-defined roles, such as nodes in Controller roles, Compute roles, and different storage role types. Each of these default roles contains a set of services that are defined in the core heat template collection.

In a default RHOSP ML2/OVN deployment, the ML2/OVN composable service runs on Controller nodes. You can optionally create a custom Networker role and run the OVN composable service on dedicated Networker nodes.

The OVN composable service **ovn-dbs** is deployed in a container called *ovn-dbs-bundle*. In a default installation **ovn-dbs** is included in the Controller role and runs on Controller nodes. Because the service is composable, you can assign it to another role, such as a Networker role.

If you assign the OVN composable service to another role, ensure that the service is co-located on the same node as the pacemaker service, which controls the OVN database containers.

Related information

- Deploying a Custom Role with ML2/OVN

2.3. DEPLOYING A CUSTOM ROLE WITH ML2/OVN

In a default RHOSP ML2/OVN deployment, the ML2/OVN composable service runs on Controller nodes. You can optionally use supported custom roles such as Networker, which runs the OVN composable services on dedicated networker nodes.

You can also generate your own custom roles.

Prerequisites

- You know how to deploy custom roles. For more information see Composable Services and Custom Roles in the Advanced Overcloud Customization guide.

Procedure

1. Log in to the undercloud host as the **stack** user and source the **stackrc** file.

```
$ source stackrc
```

2. Choose the custom roles file that is appropriate for your deployment. For example, for the Networker role, choose **Networker.yaml**. Use it directly in the deploy command if it suits your needs as-is. Or you can generate your own custom roles file that combines other custom roles files.

3. [Optional] Generate a new custom roles data file that combines one of these custom roles files with other custom roles files. Follow the instructions in Creating a roles_data file . Include the appropriate source role files depending on your deployment.

4. [Optional] To identify specific nodes for the role, you can create a specific hardware flavor and assign the flavor to specific nodes. Then use an environment file define the flavor for the role, and to specify a node count. For more information, see the example in Creating a new role .

5. Create an environment file as appropriate for your deployment. For example, for the Networker role, create a file named **neutron-ovn-dvr-ha.yaml**.

6. Include the following settings as appropriate for your deployment. For example, for the Networker role, include the following settings.

```
ControllerParameters:
   OVNCMSOptions: ""
ControllerSriovParameters:
      OVNCMSOptions: ""
NetworkerParameters:
   OVNCMSOptions: "enable-chassis-as-gw"
```

7. Deploy the overcloud. Include the environment file in your deployment command with the **-e** option. Include the custom roles data file in your deployment command with the -r option. For example: ``-r Networker.yaml` or '-r mycustomrolesfile.yaml`.

Verification steps

1. Ensure that ovn_metadata_agent is running on Controller and Networker nodes.

```
[heat-admin@controller-0 ~]$ sudo docker ps | grep ovn_metadata
```

Expect output similar to the following example.

```
a65125d9588d  undercloud-0.ctlplane.localdomain:8787/rh-osbs/rhosp13-openstack-
neutron-metadata-agent-ovn:13.1_20200813.1  kolla_start          23 hours ago  Up 21 hours
ago         ovn_metadata_agent
```

2. Ensure that Controller nodes with OVN services or dedicated Networker nodes have been configured as gateways for OVS.

```
[heat-admin@controller-0 ~]$ sudo ovs-vsctl get Open_Vswitch .
...OS::TripleO::Services::NeutronDhcpAgent: OS::Heat::None
```

Expect output similar to the following example.

```
external_ids:ovn-cms-options
   enable-chassis-as-gw
```

Additional resources

- Composable Services and Custom Roles in the Advanced Overcloud Customization guide.

2.4. HIGH AVAILABILITY WITH PACEMAKER AND DVR

You can choose one of two **ovn-dbs** profiles: the base profile, ovn-dbs-container, and the pacemaker high availability (HA) profile, ovn-dbs-container-puppet.

With the pacemaker HA profile enabled, **ovsdb-server** runs in *master-slave* mode, managed by pacemaker and the resource agent Open Cluster Framework (OCF) script. The OVN database servers start on all the Controllers, and **pacemaker** then selects one controller to serve in the master role. The instance of **ovsdb-server** that runs in *master* mode can write to the database, while all the other slave **ovsdb-server** services replicate the database locally from the *master*, and can not write to the database.

The YAML file for this profile is the **tripleo-heat-templates/environments/services-docker/neutron-ovn-dvr-ha.yaml** file. When enabled, the OVN database servers are managed by Pacemaker, and **puppet-tripleo** creates a pacemaker OCF resource named **ovn:ovndb-servers**.

The OVN database servers are started on each Controller node, and the controller owning the virtual IP address (**OVN_DBS_VIP**) runs the OVN DB servers in *master* mode. The OVN ML2 mechanism driver and **ovn-controller** then connect to the database servers using the **OVN_DBS_VIP** value. In the event of a failover, Pacemaker moves the virtual IP address (**OVN_DBS_VIP**) to another controller, and also promotes the OVN database server running on that node to *master*.

2.5. LAYER 3 HIGH AVAILABILITY WITH OVN

OVN supports Layer 3 high availability (L3 HA) without any special configuration. OVN automatically schedules the router port to all available gateway nodes that can act as an L3 gateway on the specified external network. OVN L3 HA uses the **gateway_chassis** column in the OVN **Logical_Router_Port** table. Most functionality is managed by OpenFlow rules with bundled active_passive outputs. The **ovn-controller** handles the Address Resolution Protocol (ARP) responder and router enablement and disablement. Gratuitous ARPs for FIPs and router external addresses are also periodically sent by the **ovn-controller**.

NOTE

L3HA uses OVN to balance the routers back to the original gateway nodes to avoid any nodes becoming a bottleneck.

BFD monitoring

OVN uses the Bidirectional Forwarding Detection (BFD) protocol to monitor the availability of the gateway nodes. This protocol is encapsulated on top of the Geneve tunnels established from node to node.

Each gateway node monitors all the other gateway nodes in a star topology in the deployment. Gateway nodes also monitor the compute nodes to let the gateways enable and disable routing of packets and ARP responses and announcements.

Each compute node uses BFD to monitor each gateway node and automatically steers external traffic, such as source and destination Network Address Translation (SNAT and DNAT), through the active gateway node for a given router. Compute nodes do not need to monitor other compute nodes.

NOTE

External network failures are not detected as would happen with an ML2-OVS configuration.

L3 HA for OVN supports the following failure modes:

- The gateway node becomes disconnected from the network (tunneling interface).
- **ovs-vswitchd** stops (**ovs-switchd** is responsible for BFD signaling)
- **ovn-controller** stops (**ovn-controller** removes itself as a registered node).

NOTE

This BFD monitoring mechanism only works for link failures, not for routing failures.

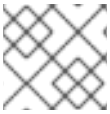
CHAPTER 3. DEPLOYING OVN WITH DIRECTOR

The following events are triggered when you deploy OVN on the Red Hat OpenStack Platform:

1. Enables the OVN ML2 plugin and generates the necessary configuration options.
2. Deploys the OVN databases and the **ovn-northd** service on the controller node(s).
3. Deploys **ovn-controller** on each Compute node.
4. Deploys **neutron-ovn-metadata-agent** on each Compute node.

3.1. DEPLOYING ML2/OVN WITH DVR

To deploy and manage distributed virtual routing (DVR) in an ML2/OVN deployment, you configure settings in heat templates and environment files.



NOTE

This procedures in this guide deploy OVN with the default DVR in an HA environment.

The default settings are provided as guidelines only. They are not expected to work in production or test environments which may require customization for network isolation, dedicated NICs, or any number of other variable factors.

The following example procedure shows how to configure a proof-of-concept deployment of ML2/OVN, HA, DVR using the typical defaults.

Procedure

1. Verify that the value for **OS::TripleO::Compute::Net::SoftwareConfig** in the **environments/services/neutron-ovn-dvr-ha.yaml** file is the same as the **OS::TripleO::Controller::Net::SoftwareConfig** value in use. This can normally be found in the network environment file used to deploy the overcloud, such as the **environments/net-multiple-nics.yaml** file. This creates the appropriate external network bridge on the Compute node.

   NOTE

   If you customize the network configuration of the Compute node, you may need to add the appropriate configuration to your custom files instead.

2. Include *environments/services/neutron-ovn-dvr-ha.yaml* as an environment file when deploying the overcloud. For example:

   ```
   $ openstack overcloud deploy \
     --templates /usr/share/openstack-tripleo-heat-templates \
     ...
    -e /usr/share/openstack-tripleo-heat-templates/environments/services/neutron-ovn-dvr-ha.yaml
   ```

3. Ensure that the Compute and Controller roles in *roles_data.yaml* include the tag *external_bridge*, and that an external network entry is added to the Compute nodes. For example:

```
- name: Compute
  description: |
    Basic Compute Node role
  CountDefault: 1
  # Create external Neutron bridge (unset if using ML2/OVS without DVR)
  tags:
    - external_bridge
  networks:
    External:
      subnet: external_subnet
...
- name: Controller
  description: |
    Controller role that has all the controller services loaded and handles
    Database, Messaging and Network functions.
  CountDefault: 1
  tags:
    - primary
    - controller
    - external_bridge
```

3.2. DEPLOYING THE OVN METADATA AGENT ON COMPUTE NODES

The OVN metadata agent is configured in the **tripleo-heat-templates/docker/services/ovn-metadata.yaml** file and included in the default Compute role through **OS::TripleO::Services::OVNMetadataAgent**. As such, the OVN metadata agent with default parameters is deployed as part of the OVN deployment. See Chapter 3, *Deploying OVN with director*.

OpenStack guest instances access the Networking metadata service available at the link-local IP address: 169.254.169.254. The **neutron-ovn-metadata-agent** has access to the host networks where the Compute metadata API exists. Each HAProxy is in a network namespace that is not able to reach the appropriate host network. HaProxy adds the necessary headers to the metadata API request and then forwards the request to the **neutron-ovn-metadata-agent** over a UNIX domain socket.

The OVN Networking service creates a unique network namespace for each virtual network that enables the metadata service. Each network accessed by the instances on the Compute node has a corresponding metadata namespace (ovnmeta-<net_uuid>).

3.2.1. Troubleshooting Metadata issues

You can use metadata namespaces for troubleshooting to access the local instances on the Compute node. To troubleshoot metadata namespace issues, run the following command as root on the Compute node:

```
# ip netns exec ovnmeta-fd706b96-a591-409e-83be-33caea824114 ssh
USER@INSTANCE_IP_ADDRESS
```

USER@INSTANCE_IP_ADDRESS is the user name and IP address for the local instance you want to troubleshoot.

3.3. DEPLOYING INTERNAL DNS WITH OVN

To use domain names instead of IP addresses on your local network for east-west traffic, use internal domain name service (DNS). With internal DNS, ovn-controller responds to DNS queries locally on the

compute node. Note that internal DNS overrides any custom DNS server specified in an instance's /etc/resolv.conf file. With internal DNS deployed, the instance's DNS queries are handled by ovn-controller instead of the custom DNS server.

Procedure

1. Enable DNS with the **NeutronPluginExtensions** parameter:

```
parameter_defaults:
  NeutronPluginExtensions: "dns"
```

2. Set the DNS domain before you deploy the overcloud:

```
  NeutronDnsDomain: "mydns-example.org"
```

3. Deploy the overcloud:

```
$ openstack overcloud deploy \
    --templates /usr/share/openstack-tripleo-heat-templates \
    ...
 -e /usr/share/openstack-tripleo-heat-templates/environments/services-docker/neutron-ovn-dvr-ha.yaml
```

CHAPTER 4. MONITORING OVN

You can use the **ovn-trace** command to monitor and troubleshoot OVN logical flows, and you can use the **ovs-ofctl dump-flows** command to monitor and troubleshoot OpenFlows.

4.1. CREATING ALIASES FOR OVN TROUBLESHOOTING COMMANDS

OVN database commands (such as ovn-nbctl show) run on the ovn_controller container. The container runs on the controller node and compute nodes. To simplify your access to the commands, create and source a script that defines aliases.

Prerequisites

- Deployment of Red Hat OpenStack Platform 13 with ML2/OVN as the mechanism driver.

Creating and using OVN database command aliases

1. Create a shell script file in the appropriate directory on the overcloud node where you want to run the ovn commands. For example, log in to the controller node as heat-admin and create the file ovn-alias.sh in the heat-admin user's ~/bin directory.

2. Save the following commands in the script file.

```
EXTERNAL_ID=\
$(sudo ovs-vsctl get open . external_ids:ovn-remote | awk -F: '{print $2}')
export NBDB=tcp:${EXTERNAL_ID}:6641
export SBDB=tcp:${EXTERNAL_ID}:6642

alias ovn-sbctl="sudo docker exec ovn_controller ovn-sbctl --db=$SBDB"
alias ovn-nbctl="sudo docker exec ovn_controller ovn-nbctl --db=$NBDB"
alias ovn-trace="sudo docker exec ovn_controller ovn-trace --db=$SBDB"
```

3. Source the script file. For example, log in to the controller node as heat-admin and run the following command.

```
# source ovn-alias.sh
```

4. Validate an alias. For example, show the northbound database.

```
ovn-nbctl show
```

Example output

```
switch 26ce22db-1795-41bd-b561-9827cbd81778 (neutron-f8e79863-6c58-43d0-8f7d-
8ec4a423e13b) (aka internal_network)
 port 1913c3ae-8475-4b60-a479-df7bcce8d9c8
    addresses: ["fa:16:3e:33:c1:fc 192.168.254.76"]
 port 1aabaee3-b944-4da2-bf0a-573215d3f3d9
    addresses: ["fa:16:3e:16:cb:ce 192.168.254.74"]
 port 7e000980-59f9-4a0f-b76a-4fdf4e86f27b
    type: localport
    addresses: ["fa:16:3e:c9:30:ed 192.168.254.2"]
```

4.2. MONITORING OVN LOGICAL FLOWS

OVN uses logical flows that are tables of flows with a priority, match, and actions. These logical flows are distributed to the **ovn-controller** running on each Compute node. You can use the **ovn-sbctl lflow-list** command on the Controller node to view the full set of logical flows, as shown in this example.

```
$ ovn-sbctl --db=tcp:172.17.1.10:6642 lflow-list
    Datapath: "sw0" (d7bf4a7b-e915-4502-8f9d-5995d33f5d10)  Pipeline: ingress
      table=0 (ls_in_port_sec_l2  ), priority=100  , match=(eth.src[40]), action=(drop;)
      table=0 (ls_in_port_sec_l2  ), priority=100  , match=(vlan.present), action=(drop;)
      table=0 (ls_in_port_sec_l2  ), priority=50   , match=(inport == "sw0-port1" && eth.src ==
{00:00:00:00:00:01}), action=(next;)
      table=0 (ls_in_port_sec_l2  ), priority=50   , match=(inport == "sw0-port2" && eth.src ==
{00:00:00:00:00:02}), action=(next;)
      table=1 (ls_in_port_sec_ip  ), priority=0    , match=(1), action=(next;)
      table=2 (ls_in_port_sec_nd  ), priority=90   , match=(inport == "sw0-port1" && eth.src ==
00:00:00:00:00:01 && arp.sha == 00:00:00:00:00:01), action=(next;)
      table=2 (ls_in_port_sec_nd  ), priority=90   , match=(inport == "sw0-port1" && eth.src ==
00:00:00:00:00:01 && ip6 && nd && ((nd.sll == 00:00:00:00:00:00 || nd.sll == 00:00:00:00:00:01) ||
((nd.tll == 00:00:00:00:00:00 || nd.tll == 00:00:00:00:00:01)))), action=(next;)
      table=2 (ls_in_port_sec_nd  ), priority=90   , match=(inport == "sw0-port2" && eth.src ==
00:00:00:00:00:02 && arp.sha == 00:00:00:00:00:02), action=(next;)
      table=2 (ls_in_port_sec_nd  ), priority=90   , match=(inport == "sw0-port2" && eth.src ==
00:00:00:00:00:02 && ip6 && nd && ((nd.sll == 00:00:00:00:00:00 || nd.sll == 00:00:00:00:00:02) ||
((nd.tll == 00:00:00:00:00:00 || nd.tll == 00:00:00:00:00:02)))), action=(next;)
      table=2 (ls_in_port_sec_nd  ), priority=80   , match=(inport == "sw0-port1" && (arp || nd)), action=
(drop;)
      table=2 (ls_in_port_sec_nd  ), priority=80   , match=(inport == "sw0-port2" && (arp || nd)), action=
(drop;)
      table=2 (ls_in_port_sec_nd  ), priority=0    , match=(1), action=(next;)
      table=3 (ls_in_pre_acl      ), priority=0    , match=(1), action=(next;)
      table=4 (ls_in_pre_lb       ), priority=0    , match=(1), action=(next;)
      table=5 (ls_in_pre_stateful ), priority=100  , match=(reg0[0] == 1), action=(ct_next;)
      table=5 (ls_in_pre_stateful ), priority=0    , match=(1), action=(next;)
      table=6 (ls_in_acl          ), priority=0    , match=(1), action=(next;)
      table=7 (ls_in_qos_mark     ), priority=0    , match=(1), action=(next;)
      table=8 (ls_in_lb           ), priority=0    , match=(1), action=(next;)
      table=9 (ls_in_stateful     ), priority=100  , match=(reg0[1] == 1), action=(ct_commit(ct_label=0/1);
next;)
      table=9 (ls_in_stateful     ), priority=100  , match=(reg0[2] == 1), action=(ct_lb;)
      table=9 (ls_in_stateful     ), priority=0    , match=(1), action=(next;)
      table=10(ls_in_arp_rsp      ), priority=0    , match=(1), action=(next;)
      table=11(ls_in_dhcp_options ), priority=0    , match=(1), action=(next;)
      table=12(ls_in_dhcp_response), priority=0    , match=(1), action=(next;)
      table=13(ls_in_l2_lkup      ), priority=100  , match=(eth.mcast), action=(outport = "_MC_flood";
output;)
      table=13(ls_in_l2_lkup      ), priority=50   , match=(eth.dst == 00:00:00:00:00:01), action=(outport
= "sw0-port1"; output;)
      table=13(ls_in_l2_lkup      ), priority=50   , match=(eth.dst == 00:00:00:00:00:02), action=(outport
= "sw0-port2"; output;)
    Datapath: "sw0" (d7bf4a7b-e915-4502-8f9d-5995d33f5d10)  Pipeline: egress
      table=0 (ls_out_pre_lb      ), priority=0    , match=(1), action=(next;)
      table=1 (ls_out_pre_acl     ), priority=0    , match=(1), action=(next;)
      table=2 (ls_out_pre_stateful), priority=100  , match=(reg0[0] == 1), action=(ct_next;)
      table=2 (ls_out_pre_stateful), priority=0    , match=(1), action=(next;)
      table=3 (ls_out_lb          ), priority=0    , match=(1), action=(next;)
```

```
    table=4 (ls_out_acl         ), priority=0    , match=(1), action=(next;)
    table=5 (ls_out_qos_mark    ), priority=0    , match=(1), action=(next;)
    table=6 (ls_out_stateful    ), priority=100  , match=(reg0[1] == 1), action=(ct_commit(ct_label=0/1);
next;)
    table=6 (ls_out_stateful    ), priority=100  , match=(reg0[2] == 1), action=(ct_lb;)
    table=6 (ls_out_stateful    ), priority=0    , match=(1), action=(next;)
    table=7 (ls_out_port_sec_ip ), priority=0    , match=(1), action=(next;)
    table=8 (ls_out_port_sec_l2 ), priority=100  , match=(eth.mcast), action=(output;)
    table=8 (ls_out_port_sec_l2 ), priority=50   , match=(outport == "sw0-port1" && eth.dst ==
{00:00:00:00:00:01}), action=(output;)
    table=8 (ls_out_port_sec_l2 ), priority=50   , match=(outport == "sw0-port2" && eth.dst ==
{00:00:00:00:00:02}), action=(output;)
```

Key differences between OVN and OpenFlow include:

- OVN ports are logical entities that reside somewhere on a network, not physical ports on a single switch.
- OVN gives each table in the pipeline a name in addition to its number. The name describes the purpose of that stage in the pipeline.
- The OVN match syntax supports complex Boolean expressions.
- The actions supported in OVN logical flows extend beyond those of OpenFlow. You can implement higher level features, such as DHCP, in the OVN logical flow syntax.

ovn-trace

The **ovn-trace** command can simulate how a packet travels through the OVN logical flows, or help you determine why a packet is dropped. Provide the **ovn-trace** command with the following parameters:

DATAPATH
: The logical switch or logical router where the simulated packet starts.

MICROFLOW
: The simulated packet, in the syntax used by the **ovn-sb** database.

This example displays the **--minimal** output option on a simulated packet and shows that the packet reaches its destination:

```
$ ovn-trace --minimal sw0 'inport == "sw0-port1" && eth.src == 00:00:00:00:00:01 && eth.dst ==
00:00:00:00:00:02'
  # reg14=0x1,vlan_tci=0x0000,dl_src=00:00:00:00:00:01,dl_dst=00:00:00:00:00:02,dl_type=0x0000
  output("sw0-port2");
```

In more detail, the **--summary** output for this same simulated packet shows the full execution pipeline:

```
$ ovn-trace --summary sw0 'inport == "sw0-port1" && eth.src == 00:00:00:00:00:01 && eth.dst ==
00:00:00:00:00:02'
# reg14=0x1,vlan_tci=0x0000,dl_src=00:00:00:00:00:01,dl_dst=00:00:00:00:00:02,dl_type=0x0000
ingress(dp="sw0", inport="sw0-port1") {
  outport = "sw0-port2";
  output;
  egress(dp="sw0", inport="sw0-port1", outport="sw0-port2") {
    output;
```

```
        /* output to "sw0-port2", type "" */;
    };
};
```

The example output shows:

- The packet enters the **sw0** network from the **sw0-port1** port and runs the ingress pipeline.
- The *outport* variable is set to **sw0-port2** indicating that the intended destination for this packet is **sw0-port2**.
- The packet is output from the ingress pipeline, which brings it to the egress pipeline for **sw0** with the *outport* variable set to **sw0-port2**.
- The output action is executed in the egress pipeline, which outputs the packet to the current value of the *outport* variable, which is **sw0-port2**.

See the **ovn-trace** man page for complete details.

4.3. MONITORING OPENFLOWS

You can use **ovs-ofctl dump-flows** command to monitor the OpenFlow flows on a logical switch in your network.

```
$ ovs-ofctl dump-flows br-int
NXST_FLOW reply (xid=0x4):
 cookie=0x0, duration=72.132s, table=0, n_packets=0, n_bytes=0, idle_age=72,
priority=10,in_port=1,dl_src=00:00:00:00:00:01 actions=resubmit(,1)
 cookie=0x0, duration=60.565s, table=0, n_packets=0, n_bytes=0, idle_age=60,
priority=10,in_port=2,dl_src=00:00:00:00:00:02 actions=resubmit(,1)
 cookie=0x0, duration=28.127s, table=0, n_packets=0, n_bytes=0, idle_age=28, priority=0
actions=drop
 cookie=0x0, duration=13.887s, table=1, n_packets=0, n_bytes=0, idle_age=13, priority=0,in_port=1
actions=output:2
 cookie=0x0, duration=4.023s, table=1, n_packets=0, n_bytes=0, idle_age=4, priority=0,in_port=2
actions=output:1
```