Red Hat OpenStack Platform 16.1

Networking with Open Virtual Network

OpenStack Networking with OVN

Last Updated: 2020-11-06

Red Hat OpenStack Platform 16.1 Networking with Open Virtual Network

OpenStack Networking with OVN

OpenStack Team
rhos-docs@redhat.com

Legal Notice

Abstract

This is an instructional guide for using OVN in OpenStack Networking Tasks.

Table of Contents

CHAPTER 1. EXPLANATION OF 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 programmatically 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.

NOTE

The minimum Open vSwitch (OVS) version required is OVS 2.9.

OVN uses Python 3.6 packages by default.

NOTE

OVN is supported only in an HA environment. We recommend that you deploy OVN with distributed virtual routing (DVR).

1.1. LIST OF COMPONENTS IN OVN ARCHITECTURE

The OVN architecture replaces the OVS Modular Layer 2 (ML2) plug-in with the OVN ML2 plug-in 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:

OVN ML2 plugin

This 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)

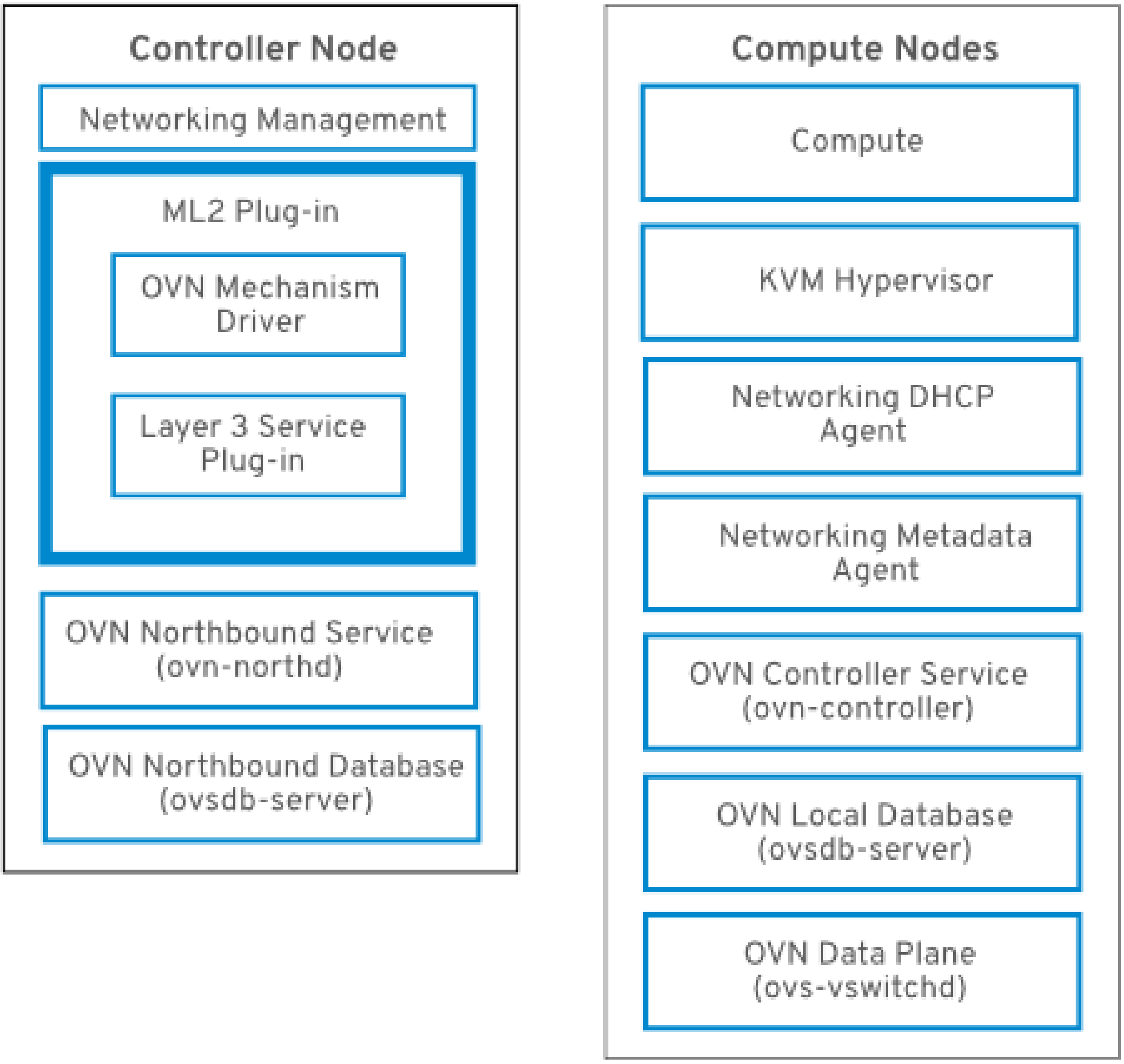
This database stores the converted logical data path flows. It typically runs on the Controller node and listens on TCP port **6642**.

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.


Controller Node
Networking Management
ML2 Plug-in
OVN Mechanism Driver
Layer 3 Service Plug-in
OVN Northbound Service (ovn-northd)
OVN Northbound Database (ovsdb-server)
Compute Nodes
Compute
KVM Hypervisor
Networking DHCP Agent
Networking Metadata Agent
OVN Controller Service (ovn-controller)
OVN Local Database (ovsdb-server)
OVN Data Plane (ovs-vswitchd)

CHAPTER 2. PLANNING YOUR OVN DEPLOYMENT

Deploy OVN in high-availability (HA) deployments only. Deploy with distributed virtual routing (DVR) enabled.

> 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.

DVR HA with OVN

Deploy OVN with DVR in an HA environment. OVN is supported only in an HA environment. 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.

2.1. THE OVN-CONTROLLER ON COMPUTE NODES

The **ovn-controller** service runs on each Compute node and connects to the OVN 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** (that 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. Below are 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

The director has a composable service for OVN named **ovn-dbs** with two profiles: the base profile and the pacemaker HA profile. The OVN northbound and southbound databases are hosted by the **ovsdb-server** service. Similarly, the **ovsdb-server** process runs alongside **ovs-vswitchd** to host the OVS database (**conf.db**).

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

2.3. HIGH AVAILABILITY WITH PACEMAKER AND DVR

In addition to the using the required HA profile, deploy OVN with the DVR to ensure the availability of networking services. With the HA profile enabled, the OVN database servers start on all the Controllers, and **pacemaker** then selects one controller to serve in the master role.

The **ovsdb-server** service does not currently support *active-active* mode. It does support HA with the *master-slave* mode, which is managed by Pacemaker using the resource agent Open Cluster Framework (OCF) script. Having **ovsdb-server** run in *master* mode allows write access to the database, while all the other slave **ovsdb-server** services replicate the database locally from the *master*, and do not allow write access.

The YAML file for this profile is the **tripleo-heat-templates/environments/services/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*.

2.4. 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. MIGRATING FROM ML2/OVS TO ML2/OVN

In a director-managed deployment, you can use the script **ovn_migration.sh** to migrate from ML2/OVS to ML2/OVN.

You can migrate from the ML2/OVS to the ML2/OVN mechanism driver in either mode:

- ovs-firewall
- ovs-hybrid

You cannot use the **ovn_migration.sh** script in deployments that are not managed by RHOSP director. To migrate to ML2/OVN in a deployment that is not managed by RHOSP director, see the file migration/README.rst and the Ansible playbook migration/migrate-to-ovn.yml.

3.1. LIMITATIONS OF THE ML2/OVN MECHANISM DRIVER

Some features available with the ML2/OVS mechanism driver are not yet supported with the ML2/OVN mechanism driver.

3.1.1. ML2/OVS features not yet supported by ML2/OVN

Feature	Notes	Track this Feature
Distributed virtual routing (DVR) with OVN on VLAN project (tenant) networks.	FIP traffic does not pass to a VLAN tenant network with ML2/OVN and DVR. DVR is enabled by default. If you need VLAN tenant networks with OVN, you can disable DVR. To disable DVR, inlcude the following lines in an environment file: ``` parameter_defaults: NeutronEnableDVR: false ```	https://bugzilla.redhat.com/show_bug.cgi?id=1704596https://bugzilla.redhat.com/show_bug.cgi?id=1766930
Fragmentation of packets on east/west UDP/ICMP traffic	In east/west traffic OVN does not yet support fragmentation of packets that are larger than the smallest MTU on the east/west path. For example: • VM1 is on Network1 with an MTU of 1300. • VM2 is on Network2 with an MTU of 1200. • A ping in either direction between VM1 and VM2 with a size of 1171 or less succeeds. A ping with a size greater than 1171 results in 100 percent packet loss.	https://bugzilla.redhat.com/show_bug.cgi?id=1891591

Feature	Notes	Track this Feature
Port Forwarding	OVN does not support port forwarding.	https://bugzilla.redhat.com/show_bug.cgi?id=1654608 https://blueprints.launchpad.net/neutron/+spec/port-forwarding
Security Groups Logging API	ML2/OVN does not provide a log file that logs security group events such as an instance trying to execute restricted operations or access restricted ports in remote servers.	https://bugzilla.redhat.com/show_bug.cgi?id=1619266
Multicast	When using ML2/OVN as the integration bridge, multicast traffic is treated as broadcast traffic. The integration bridge operates in FLOW mode, so IGMP snooping is not available. To support this, core OVN must support IGMP snooping.	https://bugzilla.redhat.com/show_bug.cgi?id=1575512
Provisioning Baremetal Machines with OVN DHCP	The built-in DHCP server on OVN presently can not provision baremetal nodes. It cannot serve DHCP for the provisioning networks. Chainbooting iPXE requires tagging (**--dhcp-match** in dnsmasq), which is not supported in the OVN DHCP server.	https://bugzilla.redhat.com/show_bug.cgi?id=1622154
OVS_DPDK	OVS_DPDK is presently not supported with OVN.	https://bugzilla.redhat.com/show_bug.cgi?id=1844615

3.1.2. Core OVN limitations

North/south routing on VF(direct) ports on VLAN tenant networks does not work with SR-IOV because the external ports are not colocated with the logical router's gateway ports. See https://bugs.launchpad.net/neutron/+bug/1875852.

3.2. ML2/OVS TO ML2/OVN MIGRATION: VALIDATED AND PROHIBITED SCENARIOS

Migration of the ML2 mechanism driver is a substantial change with broad consequences in a deployment. Red Hat continues to test and refine migration scenarios. Make sure that your scenario is validated before attempting a migration.

> **WARNING**
>
> An ML2/OVS to ML2/OVN migration alters the environment in ways that can not be reversed without a backup and restore. A failed or interrupted migration can leave the OpenStack environment inoperable. Before migrating in a production environment, test the migration in a stage environment that closely resembles your production environment, and create a reliable backup of your deployment.

3.2.1. Validated ML2/OVS to ML2/OVN migration scenarios

DVR to DVR

Start: RHOSP 16.1.1 with OVS with DVR. Geneve project (tenant) networks.
End: RHOSP 16.1.1 with OVN with DVR. Geneve project (tenant) networks.

SR-IOV and TLS-everywhere were not present in the starting environment or added during or after the migration.

Centralized routing + SR-IOV with virtual function (VF) ports only

Start: RHOSP 16.1.1 with OVS (no DVR)and SR-IOV.
End: RHOSP 16.1.1 with OVN (no DVR) and SR-IOV.

Workloads used only SR-IOV virtual function (VF) ports. SR-IOV phyical function (PF) ports caused migration failure.

3.2.2. ML2/OVS to ML2/OVN migration scenarios that failed in tests

Do not perform the ML2/OVS to ML2/OVN migration in the following scenarios until Red Hat announces that the underlying issues are resolved.

SR-IOV with physical function (PF) ports

Migration tests failed when any workload uses an SR-IOV PF port. To track progress on this issue, see https://bugzilla.redhat.com/show_bug.cgi?id=1879546.

Transport layer security everywhere (TLS-e)

Migration tests failed when the OVS deployment had TLS-e enabled. If your OVS deployment has TLS-e enabled, do not perform an ML2/OVS to ML2/OVN migration. To track progress on this issue, see https://bugzilla.redhat.com/show_bug.cgi?id=1879097 and https://bugzilla.redhat.com/show_bug.cgi?id=1872268.

OVS uses trunk ports

If your ML2/OVS deployment uses trunk ports, do not perform an ML2/OVS to ML2/OVN migration. The migration does not properly set up the trunked ports in the OVN environment. To track progress on this issue, see https://bugzilla.redhat.com/show_bug.cgi?id=1857652.

DVR with VLAN project (tenant) networks

Do not migrate to ML2/OVN with DVR and VLAN project networks. You can migrate to ML2/OVN with centralized routing. To track progress on this issue, see https://bugzilla.redhat.com/show_bug.cgi?id=1766930.

OVS traffic includes user datagram protocol (UDP) jumbo frames

If traffic in your ML2/OVS deployment includes UDP frame sizes that exceed the maximum transmission unit (MTU) of the external network, you might encounter 100 percent packet loss of the UDP traffic in your post-migration ML2/OVN environment. Support of UDP jumbo frames requires Red Hat Enterprise Linux (RHEL) version 8.2.0.4 with kernel-4.18.0-193.20.1.el8_2 or later, which is not yet available. To track availability of the required kernel, see https://bugzilla.redhat.com/show_bug.cgi?id=1860169.

3.3. PREPARING TO MIGRATE FROM ML2/OVS TO ML2/OVN

Prerequisites

- You are working with Red Hat OpenStack Platform (RHOSP) 16.0 or later.
- You have the latest openstack/neutron version in the undercloud and overcloud.
- Your RHOSP deployment is up to date. In other words, if you need to upgrade or update your OpenStack version, perform the upgrade or update first, and then perform the ML2/OVS to ML2/OVN migration.
- You have the docker-podman package installed on your undercloud and overcloud. This package might not be installed if you performed a Framework for Upgrades upgrade (FFU) from RHOSP 13 to RHOSP 16.1.1.

Procedure

In the undercloud, perform the following steps:

1. If your deployment uses VXLAN or GRE tenant networks, schedule for a 24-hour waiting period after the setup-mtu-t1 step.
   - This wait allows the VMs to catch up with the new MTU timing. During this time you may need to manually set MTUs on some instances and reboot some instances.
   - 24 hours is the time based on default configuration of 86400 seconds. The actual time depends on /var/lib/config-data/puppet-generated/neutron/etc/neutron/dhcp_agent.ini dhcp_renewal_time and /var/lib/config-data/puppet-generated/neutron/etc/neutron/neutron.conf dhcp_lease_duration parameters.
2. Install python3-networking-ovn-migration-tool.

   ```
   sudo dnf install python3-networking-ovn-migration-tool
   ```

3. Create a directory on the undercloud, and copy the Ansible playbooks:

   ```
   mkdir ~/ovn_migration
   cd ~/ovn_migration
   cp -rfp /usr/share/ansible/networking-ovn-migration/playbooks .
   ```

4. Create the **overloud-deploy-ovn.sh** script. Choose the appropriate steps based on whether your deployment was upgraded from RHOSP 13 with a fast forward upgrade (FFU).

   If your deployment was upgraded by FFU

   - Copy the file **overcloud_upgrade_prepare.sh**, which was used in the FFU, to **overcloud-deploy-ovn.sh**.

- Edit **overcloud-deploy-ovn.sh** to replace **openstack overcloud upgrade prepare** with **openstack overcloud deploy**.
- Ensure that cinder endpoints and services of type **volume** are removed. This is a workaround for a bug that impacts FFU. See https://bugzilla.redhat.com/show_bug.cgi?id=1878492.
  Run the following commands with overcloud credentials.
  - Run **openstack endpoint list | grep cinder** to verify that endpoints and services of type **volume** are not present. Only **volumev2** and **volumev3** services should be present.
  - If a service of Service Type **volume** is present, delete it. The following command deletes all endpoints of type **volume**: **openstack service delete volume**.
  - Verify the delete volume results : **openstack endpoint list | grep cinder**.

If your deployment was not upgraded by FFU

- Copy the **overcloud-deploy.sh** script to **overcloud-deploy-ovn.sh** in your **$HOME** directory.

5. Clean up the **overcloud-deploy-ovn.sh** script.
   a. Ensure the script starts with a command to source your stackrc file. For example **source ~/stackrc**.
   b. Remove any references to files specific to neutron OVS, such as **neutron-ovs-dvr.yaml**, **neutron-ovs-dpdk.yaml** and, if your deployment uses SR-IOV, **neutron-sriov.yaml**.
6. Find your migration scenario in the following list and perform the appropriate steps to customize the **openstack deploy** command in **overcloud-deploy-ovn.sh**.

   Scenario 1: DVR to DVR, compute nodes have connectivity to the external network

   - Add the following environment files to the **openstack deploy** command in overcloud-deploy-ovn.sh. Add them in the order shown.

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

   Scenario 2: Centralized routing to centralized routing (no DVR)

   - If your deployment uses SR-IOV, add **OS::TripleO::Services::OVNMetadataAgent** to the Controller role.
   - Preserve the pre-migration custom bridge mappings.
     - Run this command on the master controller to get the current bridge mappings:

```
sudo podman exec -it neutron_api crudini --get
/etc/neutron/plugins/ml2/openvswitch_agent.ini ovs bridge_mappings
```

Example output

```
datacentre:br-ex,tenant:br-isolated
```

- Create an environment file for the bridge mappings: **/home/stack/neutron_bridge_mappings.yaml**.
- Set the defaults in the environment file. For example:

```
parameter_defaults:
  ComputeParameters
    NeutronBridgeMappings: "datacentre:br-ex,tenant:br-isolated"
```

- Add the following environment files to the **openstack deploy** command in overcloud-deploy-ovn.sh. Add them in the order shown. If your environment does not use SR-IOV, omit the neutron-ovn-sriov.yaml file.

```
-e /usr/share/openstack-tripleo-heat-templates/environments/services/neutron-ovn-ha.yaml \
-e /usr/share/openstack-tripleo-heat-templates/environments/services/neutron-ovn-sriov.yaml \
-e /home/stack/ovn-extras.yaml  \
-e /home/stack/neutron_bridge_mappings.yaml
```

- Leave any custom network modifications the same as they were before migration.

Scenario 3: Centralized routing to DVR, with Geneve type driver, and compute nodes connected to external networks through br-ex

> **WARNING**
>
> If your ML2/OVS deployment uses centralized routing and VLAN project (tenant) networks, do not migrate to ML2/OVN with DVR. You can migrate to ML2/OVN with centralized routing. To track progress on this limitation, see https://bugzilla.redhat.com/show_bug.cgi?id=1704596.

- Ensure that compute nodes are connected to the external network through the br-ex bridge. For example, in an environment file such as compute-dvr.yaml, set the following:

```
type: ovs_bridge
    # Defaults to br-ex, anything else requires specific # bridge mapping entries for it to be used.
    name: bridge_name
    use_dhcp: false
    members:
     -
      type: interface
      name: nic3
      # force the MAC address of the bridge to this interface
      primary: true
```

7. Ensure that all users have execution privileges on ovn_migration.sh/ansible. The script requires execution privileges during the migration process.

```
$ chmod a+x ~/overcloud-deploy-ovn.sh
```

8. Use **export** commands to set the following migration-related environment variables. For example:

```
$ export PUBLIC_NETWORK_NAME=my-public-network
```

- STACKRC_FILE - the stackrc file in your undercloud.
  Default: ~/stackrc
- OVERCLOUDRC_FILE - the overcloudrc file in your undercloud.
  Default: ~/overcloudrc
- OVERCLOUD_OVN_DEPLOY_SCRIPT - the deployment script.
  Default: ~/overcloud-deploy-ovn.sh
- PUBLIC_NETWORK_NAME - the name of your public network.
  Default: *public*.
- IMAGE_NAME - the name or ID of the glance image to use to boot a test server.
  Default: *cirros*.

  The image is automatically downloaded during the pre-validation / post-validation process.
- VALIDATE_MIGRATION - Create migration resources to validate the migration. Before starting the migration, the migration script boots a server and validates that the server is reachable after the migration.
  Default: True.

  > **WARNING**
  >
  > Migration validation requires at least two available floating IP addresses, two networks, two subnets, two instances, and two routers as admin.
  >
  > Also, the network specified by PUBLIC_NETWORK_NAME must have available floating IP addresses, and you must be able to ping them from the undercloud.
  >
  > If your environment does not meet these requirements, set VALIDATE_MIGRATION to False.

- SERVER_USER_NAME - User name to use for logging to the migration instances.
  Default: *cirros*.
- DHCP_RENEWAL_TIME - DHCP renewal time in seconds to configure in DHCP agent configuration file.

Default: 30

9. Run **ovn_migration.sh generate-inventory** to generate the inventory file **hosts_for_migration** and the **ansible.cfg** file. Review **hosts_for_migration** for correctness.

```
$ ovn_migration.sh generate-inventory
```

10. Run **ovn_migration.sh setup-mtu-t1**. This lowers the T1 parameter of the internal neutron DHCP servers that configure the **dhcp_renewal_time** in /var/lib/config-data/puppet-generated/neutron/etc/neutron/dhcp_agent.ini in all the nodes where DHCP agent is running.

```
$ ovn_migration.sh setup-mtu-t1
```

11. If your deployment uses VLAN tenant networks, skip to step 17.

12. If your deployment uses VXLAN or GRE tenant networking, wait at least 24 hours before continuing.

13. If you have any instances with static IP assignment on VXLAN or GRE tenant networks, you must manually modify the configuration of those instances to configure the new Geneve MTU, which is the current VXLAN MTU minus 8 bytes. For instance, if the VXLAN-based MTU was 1450, change it to 1442.

14. If your instances don't honor the T1 parameter of DHCP, reboot them.

15. [Optional] Verify that the T1 parameter has propagated to existing VMs.

    - Connect to one of the compute nodes.
    - Run tcpdump over one of the VM taps attached to a tenant network.
      If T1 propagation is successful, you should see that requests happen on an interval of approximately 30 seconds:

      ```
      [heat-admin@overcloud-novacompute-0 ~]$ sudo tcpdump -i tap52e872c2-e6 port 67 or
      port 68 -n
      tcpdump: verbose output suppressed, use -v or -vv for full protocol decode
      listening on tap52e872c2-e6, link-type EN10MB (Ethernet), capture size 262144 bytes
      13:17:28.954675 IP 192.168.99.5.bootpc > 192.168.99.3.bootps: BOOTP/DHCP,
      Request from fa:16:3e:6b:41:3d, length 300
      13:17:28.961321 IP 192.168.99.3.bootps > 192.168.99.5.bootpc: BOOTP/DHCP, Reply,
      length 355
      13:17:56.241156 IP 192.168.99.5.bootpc > 192.168.99.3.bootps: BOOTP/DHCP,
      Request from fa:16:3e:6b:41:3d, length 30013:17:56.249899 IP 192.168.99.3.bootps >
      192.168.99.5.bootpc: BOOTP/DHCP, Reply, length 355
      ```

      NOTE This verification is not possible with cirros VMs. The cirros udhcpc implementation does not obey DHCP option 58 (T1). Try this verification on a port that belongs to a full Linux VM. Red Hat recommends that you check all the different types of workloads that your system runs (Windows, different flavors of Linux, etc..).

16. Lower the MTU of the:pre-migration VXLAN and GRE networks:

```
$ ovn_migration.sh reduce-mtu
```

This step reduces the MTU network by network and tags the completed network with adapted_mtu. The tool ignores non-VXLAN/GRE networks, so if you use VLAN for tenant networks, this step is not expected to change any values.

17. Make Tripleo prepare the new container images for OVN.
    If your deployment did not have a containers-prepare-parameter.yaml, you can create one with the following command:

```
$ test -f $HOME/containers-prepare-parameter.yaml || sudo openstack tripleo container
image prepare default \
--output-env-file $HOME/containers-prepare-parameter.yaml
```

    If you had to create the file, verify that it is present at the end of your $HOME/overcloud-deploy-ovn.sh and $HOME/overcloud-deploy.sh

    Change the neutron_driver in the containers-prepare-parameter.yaml file to ovn:

```
$ sed -i -E 's/neutron_driver:([ ]\w+)/neutron_driver: ovn/' $HOME/containers-prepare-
parameter.yaml
```

    [Optional] Verify the changes to the neutron_driver:

```
$ grep neutron_driver $HOME/containers-prepare-parameter.yaml
neutron_driver: ovn
```

    Update the images:

```
$ sudo openstack tripleo container image prepare \
--environment-file /home/stack/containers-prepare-parameter.yaml
```

> **NOTE**
>
> Provide the full path to your containers-prepare-parameter.yaml file. Otherwise, the command completes very quickly without updating the images or providing an error message.

    TripleO validates the containers and pushes them to your local registry.

3.4. MIGRATING FROM ML2/OVS TO ML2/OVN

1. Run **ovn_migration.sh start-migration** to kick-start the migration process.

```
$ ovn_migration.sh start-migration
```

    The script performs the following actions.

    - Creates pre-migration resources (network and VM) to validate existing deployment and final migration.
    - Updates the overcloud stack to deploy OVN alongside reference implementation services using the temporary bridge br-migration instead of br-int.
    - Starts the migration process, which includes the following actions:

  - Generates the OVN north db by running neutron-ovn-db-sync util.
  - Clones the existing resources from br-int to br-migration, to allow ovn to find the same resource UUIDS over br-migration.
  - Re-assigns ovn-controller to br-int instead of br-migration.
  - Cleans up network namespaces (fip, snat, qrouter, qdhcp).
  - Removes any unnecessary patch ports on br-int.
  - Removes br-tun and br-migration ovs bridges.
  - Deletes qr-**, ha-** and qg-* ports from br-int (via neutron netns cleanup).
- Deletes neutron agents and neutron HA internal networks from the database via API.
- Validates connectivity on pre-migration resources.
- Deletes pre-migration resources.
- Creates post-migration resources.
- Validates connectivity on post-migration resources.
- Cleans up post-migration resources.
- Re-runs the deployment tool to update OVN on br-int.

CHAPTER 4. 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.

4.1. DEPLOYING OVN WITH DVR

NOTE

This guide deploys OVN with the default DVR in an HA environment.

To deploy OVN with DVR in an HA environment:

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 in use when deploying the overcloud, for example, in the **environments/net-multiple-nics.yaml** file. This creates the appropriate external network bridge on the Compute node.

   NOTE

   If customizations have been made to the network configuration of the Compute node, it may be necessary to add the appropriate configuration to those files instead.

2. Configure a Networking port for the Compute node on the external network by modifying **OS::TripleO::Compute::Ports::ExternalPort** to an appropriate value, such as **OS::TripleO::Compute::Ports::ExternalPort: ../network/ports/external.yaml**

3. 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
```

For production environments (or test environments that require special customization, such as network isolation or dedicated NICs, you can use the example environments as a guide. Pay special attention to the bridge mapping type parameters used, for example, by OVS and any reference to external facing bridges.

4.2. DEPLOYING THE OVN METADATA AGENT ON COMPUTE NODES

The OVN metadata agent is configured in the **tripleo-heat-templates/deployment/ovn/ovn-metadata-container-puppet.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 4, *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>).

4.2.1. 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.

4.3. DEPLOYING INTERNAL DNS WITH OVN

To use domain names instead of IP addresses on your local nentwork 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 itnrnal 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/neutron-ovn-dvr-ha.yaml
   ```

CHAPTER 5. 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.

5.1. 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.

5.2. 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
```