



Red Hat OpenStack Platform 16.0

Networking with Open Virtual Network

OpenStack Networking with OVN

Red Hat OpenStack Platform 16.0 Networking with Open Virtual Network

OpenStack Networking with OVN

OpenStack Team
rhos-docs@redhat.com

Legal Notice

Copyright © 2020 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, the Red Hat 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 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

A Cookbook for using OVN for OpenStack Networking Tasks.

Table of Contents

CHAPTER 1. OPEN VIRTUAL NETWORK (OVN)	3
1.1. QUICK STEPS: DEPLOYING CONTAINERIZED OVN ON THE OVERCLOUD	3
1.2. OVN ARCHITECTURE	3
CHAPTER 2. PLANNING YOUR OVN DEPLOYMENT	5
2.1. THE OVN-CONTROLLER ON COMPUTE NODES	5
2.2. THE OVN COMPOSABLE SERVICE	5
2.3. HIGH AVAILABILITY WITH PACEMAKER AND DVR	5
2.4. LAYER 3 HIGH AVAILABILITY WITH OVN	6
CHAPTER 3. MIGRATING FROM ML2/OVS TO ML2/OVN	8
3.1. OVERVIEW	8
3.2. BEFORE YOU BEGIN	8
3.3. PREPARING FOR THE MIGRATION	8
3.4. PERFORMING THE MIGRATION	13
CHAPTER 4. DEPLOYING OVN WITH DIRECTOR	14
4.1. DEPLOYING OVN WITH DVR	14
4.2. DEPLOYING THE OVN METADATA AGENT ON COMPUTE NODES	14
4.2.1. Troubleshooting Metadata issues	15
4.3. DEPLOYING INTERNAL DNS WITH OVN	15
CHAPTER 5. MONITORING OVN	16
5.1. MONITORING OVN LOGICAL FLOWS	16
5.2. MONITORING OPENFLOWS	18

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-agnostic support for the full OpenStack Networking API. OVN allows you to 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.

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



NOTE

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

1.1. QUICK STEPS: DEPLOYING CONTAINERIZED OVN ON THE OVERCLOUD

If you are already familiar with OVN, you can use this quick step to deploy OVN with DVR in an HA configuration on 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
  ....
```

1.2. OVN ARCHITECTURE

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

The OVN architecture consists of the following components and services:

OVN ML2 plugin

Translates the OpenStack-specific networking configuration into the platform-agnostic OVN logical networking configuration. This plugin typically runs on the Controller node.

OVN Northbound (NB) database (ovn-nb)

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

OVN Northbound service (ovn-northd)

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

OVN Southbound (SB) database (ovn-sb)

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

OVN controller (ovn-controller)

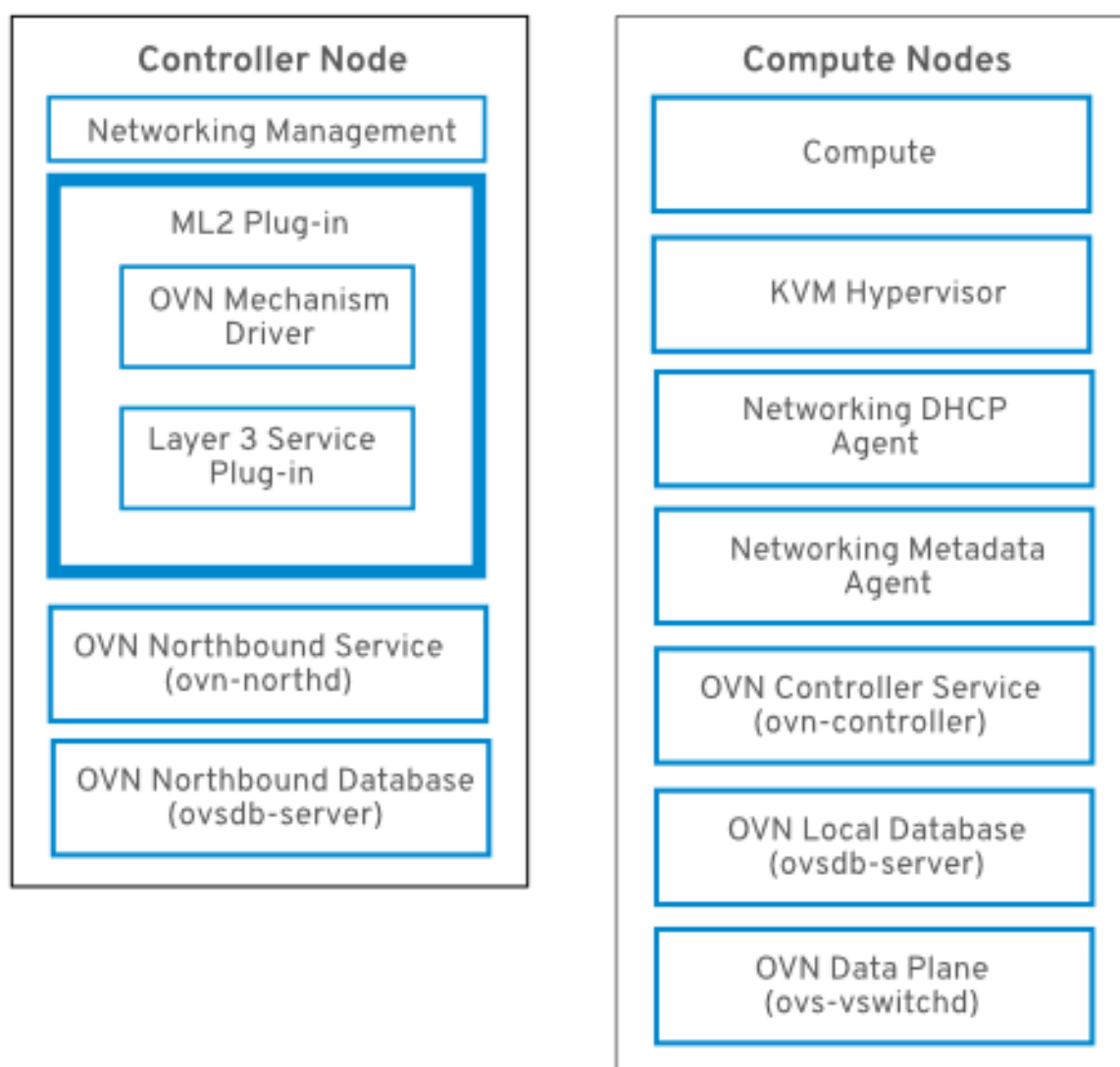
Connects to the OVN SB database and acts as the open vSwitch controller to control and monitor network traffic. Runs on all Compute and gateway nodes where

OS::TripleO::Services::OVNController is defined.

OVN metadata agent (ovn-metadata-agent)

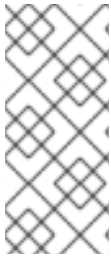
Spawns the **haproxy** instances for managing the OVS interfaces, network namespaces and HAProxy processes used to proxy metadata API requests. Runs on all Compute and gateway nodes where

OS::TripleO::Services::OVNMetadataAgent is defined.



CHAPTER 2. PLANNING YOUR OVN DEPLOYMENT

Deploy OVN in HA deployments only. We recommend you 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. 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 required HA profile, Red Hat recommends that you deploy OVN with 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

This section details an in-place migration strategy from ML2/OVS to ML2/OVN in either ovs-firewall or ovs-hybrid mode for a TripleO OpenStack deployment.

For non TripleO deployments, see the file **migration/README.rst** and the Ansible playbook **migration/migrate-to-ovn.yml**.

3.1. OVERVIEW

The migration process is orchestrated through the shell script `ovn_migration.sh`, which is provided with `networking-ovn`.

The administrator uses `ovn_migration.sh` to perform readiness steps and migration from the undercloud node. The readiness steps, such as host inventory production, DHCP and MTU adjustments, prepare the environment for the procedure.

Subsequent steps start the migration via Ansible.

Plan for a 24-hour wait after the `setup-mtu-t1` step to allow VMs to catch up with the new MTU timing. The default neutron ML2/OVS configuration has a `dhcp_lease_duration` of 86400 seconds (24h).

Also, if there are instances using static IP assignment, the administrator should plan to lower the MTU of those instances to the new value of 8 bytes less than the ML2/OVS (VXLAN) MTU value. For example, the typical 1500 MTU network value that makes VXLAN tenant networks use 1450 bytes of MTU must change to 1442 under Geneve. Or under the same overlay network, a GRE encapsulated tenant network would use a 1458 MTU, but again a 1442 MTU for Geneve.

If there are instances which use DHCP but don't support lease update during the T1 period the administrator must reboot them to ensure that MTU is updated inside those instances.

3.2. BEFORE YOU BEGIN

Ensure that you have updated to the latest `openstack/neutron` version in the undercloud/overcloud.

3.3. PREPARING FOR THE MIGRATION

In the undercloud, perform the following steps:

1. Install `python3-networking-ovn-migration-tool`.

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

2. 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 .
```

3. Create the file `$HOME/ovn-migration.yaml` with the following contents to set the `ForceNeutronDriverUpdate` parameter:

```
parameter_defaults:
  ForceNeutronDriverUpdate: true
```

**NOTE**

The `ovn-migration.yaml` file creation step is part of a workaround for the known issue described in [BZ1769880](#). In a future release, this step and the file will not be required.

4. In your **`overcloud-deploy.sh`** script, remove any references to files specific to neutron ovs, such as **`neutron-ovs-dvr.yaml`** and **`neutron-ovs-dpdk.yaml`**.
5. Create the **`overcloud-deploy-ovn.sh`** script in your **`$HOME`**. This script must source your `stackrc` file, and then execute an **`openstack overcloud overcloud deploy`** with your original deployment parameters plus the following environment files. Note that **`ovn-extras.yaml`** is automatically created by the migration tool.

When your network topology is DVR and your compute nodes have connectivity to the external network, add the files to the command in the order shown in the following example.

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

When your compute nodes don't have external connectivity and you don't use DVR, add the files as shown in the following example.

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

6. After the `-e $HOME/ovn-extras.yaml` argument, add the following argument to the command:

```
-e $HOME/ovn-migration.yaml
```

**NOTE**

The `ovn-migration.yaml` argument creation step is part of a workaround for the known issue described in [BZ1769880](#). In a future release, this step and the command argument will not be required.

7. Ensure that all users have execution privileges on the script, because it is called by `ovn_migration.sh/ansible` during the migration process.

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

8. Ensure that the **`overcloud-deploy-ovn.sh`** script does not contain any references to files specific to neutron ovs, such as **`neutron-ovs-dvr.yaml`** and **`neutron-ovs-dpdk.yaml`**.
9. To configure the parameters of your migration you can set the environment variables that will be used by `ovn_migration.sh`. You can skip setting any values matching the defaults.
 - `STACKRC_FILE` - must point to your `stackrc` file in your undercloud.
Default: `~/stackrc`
 - `OVERCLOUDRC_FILE` - must point to your `overcloudrc` file in your undercloud.
Default: `~/overcloudrc`

- **OVERCLOUD_OVN_DEPLOY_SCRIPT** - must point to the deployment script.
Default: `~/overcloud-deploy-ovn.sh`
- **PUBLIC_NETWORK_NAME** - Name of your public network.
Default: `public`.

To support migration validation, this network must have available floating IP addresses, and those floating IP addresses must be pingable from the undercloud. If that is not possible, configure **VALIDATE_MIGRATION** to `False`.

- **IMAGE_NAME** - Name/ID of the glance image to use for booting 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. The migration script, before starting the migration, boot a server and validates that the server is reachable after the migration.
Default: `True`.
- **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`



WARNING

VALIDATE_MIGRATION requires enough quota (2 available floating IP addresses, 2 networks, 2 subnets, 2 instances, and 2 routers as admin).

For example:

```
$ export PUBLIC_NETWORK_NAME=my-public-network
$ ovn_migration.sh .....
```

10. 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
```

11. Run **ovn_migration.sh setup-mtu-t1**. This lowers the T1 parameter of the internal neutron DHCP servers configuring 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
```

12. If you are using VXLAN or GRE tenant networking, wait at least 24 hours before continuing. This allows VMs to catch up with the new MTU timing of the next step.



WARNING

If you are using VXLAN or GRE networks, this 24-hour wait step is critical. If you are using VLAN tenant networks you can proceed to the next step without delay.



WARNING

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. If your instances don't honor the T1 parameter of DHCP, they must be rebooted.



NOTE

24 hours is the time based on default configuration. It actually 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. (It defaults to 86400 seconds.)



NOTE

Migrating a deployment which uses VLAN for tenant/project networks is not recommended at this time because of a bug in core ovn. See <https://mail.openvswitch.org/pipermail/ovs-dev/2018-May/347594.html>

One way to verify that the T1 parameter has propagated to existing VMs is to connect to one of the compute nodes and run **tcpdump** over one of the VM taps attached to a tenant network. If T1 propagation was a success, 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 300
13:17:56.249899 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 udhcpd implementation does not obey DHCP option 58 (T1). Try this verification on a port that belongs to a full Linux VM. It is recommended that you check all the different types of workloads your system runs (Windows, different flavors of Linux, etc..).

13. 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.

14. 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:

```
$ test -f $HOME/containers-prepare-parameter.yaml || 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 included 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
```

You can verify with:

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

Then 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. PERFORMING THE MIGRATION

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 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 deploy internal DNS with OVN:

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=(output = "_MC_flood";
output;)
  table=13(ls_in_l2_lkup ), priority=50 , match=(eth.dst == 00:00:00:00:00:01), action=(output
= "sw0-port1"; output;)
  table=13(ls_in_l2_lkup ), priority=50 , match=(eth.dst == 00:00:00:00:00:02), action=(output
= "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,d_l_src=00:00:00:00:00:01,d_l_dst=00:00:00:00:00:02,d_l_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,d_l_src=00:00:00:00:00:01,d_l_dst=00:00:00:00:00:02,d_l_type=0x0000

```

```

ingress(dp="sw0", inport="sw0-port1") {
    output = "sw0-port2";
    output;
    egress(dp="sw0", inport="sw0-port1", output="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 *output* 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 *output* variable set to **sw0-port2**.
- The output action is executed in the egress pipeline, which outputs the packet to the current value of the *output* 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

```