Configuring network functions virtualization

Plan and configure a Network Functions Virtualization (NFV) OpenStack Deployment
Plan and configure a Network Functions Virtualization (NFV) OpenStack Deployment

OpenStack Team
rhos-docs@redhat.com
Abstract

This guide contains important planning information and describes the configuration procedures for single root input/output virtualization (SR-IOV) and dataplane development kit (DPDK) for network functions virtualization infrastructure (NFVI) in your Red Hat OpenStack Platform deployment.
Table of Contents

MAKING OPEN SOURCE MORE INCLUSIVE ......................................................... 5

PROVIDING FEEDBACK ON RED HAT DOCUMENTATION .................................. 6

CHAPTER 1. OVERVIEW OF NFV ................................................................. 7

CHAPTER 2. HARDWARE REQUIREMENTS ....................................................... 8
  2.1. TESTED NICS ......................................................................................... 8
  2.2. TROUBLESHOOTING HARDWARE OFFLOAD ....................................... 8
  2.3. DISCOVERING YOUR NUMA NODE TOPOLOGY .................................. 9
  2.4. RETRIEVING HARDWARE INTROSPECTION DETAILS ....................... 9
  2.5. NFV BIOS SETTINGS ........................................................................... 13

CHAPTER 3. SOFTWARE REQUIREMENTS ...................................................... 15
  3.1. REGISTERING AND ENABLING REPOSITORIES .................................. 15
  3.2. SUPPORTED CONFIGURATIONS FOR NFV DEPLOYMENTS ................. 16
    3.2.1. Deploying RHOSP with the OVS mechanism driver ......................... 16
    3.2.2. Deploying OVN with OVS-DPDK and SR-IOV ............................... 17
    3.2.3. Deploying OVN with OVS TC Flower offload ............................... 18
    3.3. SUPPORTED DRIVERS ...................................................................... 20
    3.4. COMPATIBILITY WITH THIRD-PARTY SOFTWARE ............................. 20

CHAPTER 4. NETWORK CONSIDERATIONS .................................................. 21

CHAPTER 5. PLANNING AN SR-IOV DEPLOYMENT ......................................... 22
  5.1. HARDWARE PARTITIONING FOR AN SR-IOV DEPLOYMENT ................ 22
  5.2. TOPOLOGY OF AN NFV SR-IOV DEPLOYMENT .................................. 22
    5.2.1. Topology for NFV SR-IOV without HCI ..................................... 23

CHAPTER 6. DEPLOYING SR-IOV TECHNOLOGIES ........................................ 25
  6.1. PREREQUISITES FOR DEPLOYING SR-IOV TECHNOLOGIES ............... 25
  6.2. CONFIGURING SR-IOV ....................................................................... 25
  6.3. CONFIGURING NIC PARTITIONING ................................................... 28
  6.4. EXAMPLE NIC PARTITIONING CONFIGURATIONS ............................. 34
  6.5. CONFIGURING OVS HARDWARE OFFLOAD ...................................... 36
    6.5.1. Verifying OVS hardware offload ............................................... 38
  6.6. TUNING EXAMPLES FOR OVS HARDWARE OFFLOAD ....................... 38
  6.7. COMPONENTS OF OVS HARDWARE OFFLOAD .................................... 39
  6.8. TROUBLESHOOTING OVS HARDWARE OFFLOAD ............................. 41
  6.9. DEBUGGING HW OFFLOAD FLOW ..................................................... 45
  6.10. DEPLOYING AN INSTANCE FOR SR-IOV .......................................... 46
  6.11. CREATING HOST AGGREGATES ..................................................... 47

CHAPTER 7. PLANNING YOUR OVS-DPDK DEPLOYMENT ............................... 49
  7.1. OVS-DPDK WITH CPU PARTITIONING AND NUMA TOPOLOGY ............ 49
  7.2. CALCULATING OVS-DPDK PARAMETERS ......................................... 50
    7.2.1. CPU parameters ....................................................................... 50
    7.2.2. Memory parameters ............................................................... 51
    7.2.3. Networking parameters ........................................................... 53
    7.2.4. Other parameters .................................................................... 54
    7.2.5. Instance extra specifications .................................................... 54
  7.3. TWO NUMA NODE EXAMPLE OVS-DPDK DEPLOYMENT ................... 55
  7.4. TOPOLOGY OF AN NFV OVS-DPDK DEPLOYMENT ............................. 56
<table>
<thead>
<tr>
<th>Section</th>
<th>Page</th>
</tr>
</thead>
<tbody>
<tr>
<td>11.1. CONFIGURING ROLES DATA</td>
<td>93</td>
</tr>
<tr>
<td>11.2. CONFIGURING OVS-DPDK PARAMETERS</td>
<td>93</td>
</tr>
<tr>
<td>11.3. CONFIGURING THE CONTROLLER NODE</td>
<td>95</td>
</tr>
<tr>
<td>11.4. CONFIGURING THE COMPUTE NODE FOR DPDK AND SR-IOV</td>
<td>96</td>
</tr>
<tr>
<td>11.5. DEPLOYING THE OVERCLOUD</td>
<td>97</td>
</tr>
<tr>
<td>CHAPTER 12. UPGRADE RED HAT OPENSTACK PLATFORM WITH NFV</td>
<td>98</td>
</tr>
<tr>
<td>CHAPTER 13. NFV PERFORMANCE</td>
<td>99</td>
</tr>
<tr>
<td>CHAPTER 14. FINDING MORE INFORMATION</td>
<td>100</td>
</tr>
<tr>
<td>CHAPTER 15. SAMPLE DPDK SRIOV YAML FILES</td>
<td>101</td>
</tr>
<tr>
<td>15.1. SAMPLE VXLAN DPDK SRIOV YAML FILES</td>
<td></td>
</tr>
<tr>
<td>15.1.1. roles_data.yaml</td>
<td>101</td>
</tr>
<tr>
<td>15.1.2. network-environment-overrides.yaml</td>
<td>106</td>
</tr>
<tr>
<td>15.1.3. controller.yaml</td>
<td>107</td>
</tr>
<tr>
<td>15.1.4. compute-ovs-dpdk.yaml</td>
<td>109</td>
</tr>
<tr>
<td>15.1.5. overcloud_deploy.sh</td>
<td>111</td>
</tr>
</tbody>
</table>
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.
PROVIDING FEEDBACK ON RED HAT DOCUMENTATION

We appreciate your input on our documentation. Tell us how we can make it better.

You are viewing a beta version of documentation, and feedback is temporarily suspended. If you see any inaccuracies that are published, check back after the general availability (GA) of Red Hat OpenStack Platform and documentation release. If the issue persists, tell us how we can make it better. You can use the following steps after GA to provide feedback:

Using the Direct Documentation Feedback (DDF) function

Use the Add Feedback DDF function for direct comments on specific sentences, paragraphs, or code blocks.

1. View the documentation in the Multi-page HTML format.
2. Ensure that you see the Feedback button in the upper right corner of the document.
3. Highlight the part of text that you want to comment on.
4. Click Add Feedback.
5. Complete the Add Feedback field with your comments.
6. Optional: Add your email address so that the documentation team can contact you for clarification on your issue.
7. Click Submit.
CHAPTER 1. OVERVIEW OF NFV

Network Functions Virtualization (NFV) is a software solution that virtualizes a network function, such as a network switch, on general purpose, cloud-based infrastructure. NFV allows the Communication Service Provider to move away from traditional or proprietary hardware.

For a high-level overview of NFV concepts, see the Introduction to network functions virtualization.

NOTE

OVS-DPDK and SR-IOV configuration depends on your hardware and topology. This guide provides examples for CPU assignments, memory allocation, and NIC configurations that might vary from your topology and use case.

Use Red Hat OpenStack Platform director to isolate specific network types, for example, external, project, internal API, and so on. You can deploy a network on a single network interface, or distributed over a multiple-host network interface. With Open vSwitch you can create bonds by assigning multiple interfaces to a single bridge. Configure network isolation in a Red Hat OpenStack Platform installation with template files. If you do not provide template files, the service networks deploy on the provisioning network. There are two types of template configuration files:

- **network-environment.yaml** - this file contains network details, such as subnets and IP address ranges, for the overcloud nodes. This file also contains the different settings that override the default parameter values for various scenarios.

- **Host network templates**, for example, **compute.yaml** and **controller.yaml** - define the network interface configuration for the overcloud nodes. The values of the network details are provided by the **network-environment.yaml** file.

These heat template files are located at `/usr/share/openstack-tripleo-heat-templates/` on the undercloud node.

The Hardware requirements and Software requirements sections provide more details on how to plan and configure the heat template files for NFV using the Red Hat OpenStack Platform director.

NOTE

You can edit YAML files to configure NFV. For an introduction to the YAML file format, see: YAML in a Nutshell.
CHAPTER 2. HARDWARE REQUIREMENTS

This section describes the hardware requirements for NFV.

For a complete list of the certified hardware for Red Hat OpenStack Platform, see Red Hat OpenStack Platform certified hardware.

2.1. TESTED NICS

For a list of tested NICs for NFV, see the Red Hat Knowledgebase solution Network Adapter Fast Datapath Feature Support Matrix.

If you configure OVS-DPDK on Mellanox ConnectX-4 or ConnectX-5 network interfaces, you must set the corresponding kernel driver in the compute-ovs-dpdk.yaml file:

```
members
  - type: ovs_dpdk_port
    name: dpdk0
    driver: mlx5_core
    members:
      - type: interface
        name: enp3s0f0
```

2.2. TROUBLESHOOTING HARDWARE OFFLOAD

In a Red Hat OpenStack Platform(RHOSP) 17.1-Beta deployment, OVS Hardware Offload might not offload flows for VMs with switchdev-capable ports and Mellanox ConnectX5 NICs. To troubleshoot and configure offload flows in this scenario, disable the ESWITCH_IPV4_TTL_MODIFY_ENABLE Mellanox firmware parameter. For more troubleshooting information about OVS Hardware Offload in RHOSP 17.1-Beta, see the Red Hat Knowledgebase solution OVS Hardware Offload with Mellanox NIC in OpenStack Platform 16.2.

Procedure

1. Log in to the Compute nodes in your RHOSP deployment that have Mellanox NICs that you want to configure.

2. Use the mstflint utility to query the ESWITCH_IPV4_TTL_MODIFY_ENABLE Mellanox firmware parameter.

   ```
   [root@compute-1 ~]# yum install -y mstflint
   [root@compute-1 ~]# mstconfig -d <PF PCI BDF> q
   ESWITCH_IPV4_TTL_MODIFY_ENABLE
   ```

3. If the ESWITCH_IPV4_TTL_MODIFY_ENABLE parameter is enabled and set to 1, then set the value to 0 to disable it.

   ```
   [root@compute-1 ~]# mstconfig -d <PF PCI BDF> s
   ESWITCH_IPV4_TTL_MODIFY_ENABLE=0
   ```

4. Reboot the node.
2.3. DISCOVERING YOUR NUMA NODE TOPOLOGY

When you plan your deployment, you must understand the NUMA topology of your Compute node to partition the CPU and memory resources for optimum performance. To determine the NUMA information, perform one of the following tasks:

- Enable hardware introspection to retrieve this information from bare-metal nodes.
- Log on to each bare-metal node to manually collect the information.

**NOTE**

You must install and configure the undercloud before you can retrieve NUMA information through hardware introspection. For more information about undercloud configuration, see: Installing and managing Red Hat OpenStack Platform with director Guide.

2.4. RETRIEVING HARDWARE INTROSPECTION DETAILS

The Bare Metal service hardware-inspection-extras feature is enabled by default, and you can use it to retrieve hardware details for overcloud configuration. For more information about the `inspection_extras` parameter in the `undercloud.conf` file, see Director configuration parameters.

For example, the `numa_topology` collector is part of the hardware-inspection extras and includes the following information for each NUMA node:

- RAM (in kilobytes)
- Physical CPU cores and their sibling threads
- NICs associated with the NUMA node

**Procedure**

- To retrieve the information listed above, substitute `<UUID>` with the UUID of the bare-metal node to complete the following command:

  ```
  # openstack baremetal introspection data save <UUID> | jq .numa_topology
  ```

  The following example shows the retrieved NUMA information for a bare-metal node:

  ```json
  {
    "cpus": [
      {
        "cpu": 1,
        "thread_siblings": [1, 17],
        "numa_node": 0
      },
      {
        "cpu": 2,
        "thread_siblings": [10, 26]
      }
    ]
  }
  ```
```json
[
  {
    "cpu": 5,
    "thread_siblings": [
      5,
      21
    ],
    "numa_node": 0
  },
  {
    "cpu": 4,
    "thread_siblings": [
      12,
      28
    ],
    "numa_node": 1
  },
  {
    "cpu": 4,
    "thread_siblings": [
      4,
      20
    ],
    "numa_node": 0
  },
  {
    "cpu": 0,
    "thread_siblings": [
      8,
      24
    ],
    "numa_node": 1
  },
  {
    "cpu": 6,
    "thread_siblings": [
      14,
      30
    ],
    "numa_node": 1
  },
  {
    "cpu": 3,
    "thread_siblings": [
      3,
      19
    ],
    "numa_node": 0
  },
  {
    "cpu": 2,
    "thread_siblings": [
      2,
      18
    ],
    "numa_node": 1
  }
]
```
2.5. NFV BIOS SETTINGS

The following table describes the required BIOS settings for NFV:

NOTE

You must enable SR-IOV global and NIC settings in the BIOS, or your Red Hat OpenStack Platform (RHOSP) deployment with SR-IOV Compute nodes will fail.

Table 2.1. BIOS Settings

<table>
<thead>
<tr>
<th>Parameter</th>
<th>Setting</th>
</tr>
</thead>
<tbody>
<tr>
<td>C3 Power State</td>
<td>Disabled.</td>
</tr>
<tr>
<td>C6 Power State</td>
<td>Disabled.</td>
</tr>
<tr>
<td>MLC Streamer</td>
<td>Enabled.</td>
</tr>
<tr>
<td>MLC Spatial Prefetcher</td>
<td>Enabled.</td>
</tr>
<tr>
<td>DCU Data Prefetcher</td>
<td>Enabled.</td>
</tr>
<tr>
<td>DCA</td>
<td>Enabled.</td>
</tr>
<tr>
<td>CPU Power and Performance</td>
<td>Performance.</td>
</tr>
<tr>
<td>Memory RAS and Performance Config</td>
<td>NUMA Optimized</td>
</tr>
<tr>
<td>→ NUMA Optimized</td>
<td>Enabled.</td>
</tr>
<tr>
<td>Turbo Boost</td>
<td>Disabled in NFV deployments that require deterministic performance. Enabled in all other scenarios.</td>
</tr>
<tr>
<td>VT-d</td>
<td>Enabled for Intel cards if VFIO functionality is needed.</td>
</tr>
<tr>
<td>NUMA memory interleave</td>
<td>Disabled.</td>
</tr>
</tbody>
</table>

On processors that use the intel_idle driver, Red Hat Enterprise Linux can ignore BIOS settings and re-enable the processor C-state.

You can disable intel_idle and instead use the acpi_idle driver by specifying the key-value pair intel_idle.max_cstate=0 on the kernel boot command line.

Confirm that the processor is using the acpi_idle driver by checking the contents of current_driver.
# cat /sys/devices/system/cpu/cpuidle/current_driver
acpi_idle

NOTE

You will experience some latency after changing drivers, because it takes time for the Tuned daemon to start. However, after Tuned loads, the processor does not use the deeper C-state.
CHAPTER 3. SOFTWARE REQUIREMENTS

This section describes the supported configurations and drivers, and subscription details necessary for NFV.

3.1. REGISTERING AND ENABLING REPOSITORIES

To install Red Hat OpenStack Platform, you must register Red Hat OpenStack Platform director using the Red Hat Subscription Manager, and subscribe to the required channels. For more information about registering and updating your undercloud, see Registering your system.

Procedure

1. Register your system with the Content Delivery Network, entering your Customer Portal user name and password when prompted.
   
   ```
   [stack@director ~]$ sudo subscription-manager register
   ```

2. Determine the entitlement pool ID for Red Hat OpenStack Platform director, for example {Pool ID} from the following command and output:
   
   ```
   [stack@director ~]$ sudo subscription-manager list --available --all --matches="Red Hat OpenStack"
   
   Subscription Name:   Name of SKU
   Provides:            Red Hat Single Sign-On
                         Red Hat Enterprise Linux Workstation
                         Red Hat CloudForms
                         Red Hat OpenStack
                         Red Hat Software Collections (for RHEL Workstation)
   SKU:                 SKU-Number
   Contract:            Contract-Number
   Pool ID:             {Pool-ID}-123456
   Provides Management: Yes
   Available:           1
   Suggested:           1
   Service Level:       Support-level
   Service Type:        Service-Type
   Subscription Type:   Sub-type
   Ends:                End-date
   System Type:         Physical
   ```

3. Include the Pool ID value in the following command to attach the Red Hat OpenStack Platform 16.1 entitlement.
   
   ```
   [stack@director ~]$ sudo subscription-manager attach --pool={Pool-ID}-123456
   ```

4. Disable the default repositories.
   
   ```
   subscription-manager repos --disable=* 
   ```

5. Enable the required repositories for Red Hat OpenStack Platform with NFV.
   
   ```
   $ sudo subscription-manager repos --enable=rhel-8-for-x86_64-baseos-eus-rpms \
   ```
6. Update your system so you have the latest base system packages.

```
[stack@director ~]$ sudo dnf update -y
[stack@director ~]$ sudo reboot
```

**NOTE**

To register your overcloud nodes, see [Ansible Based Registration](#).

### 3.2. SUPPORTED CONFIGURATIONS FOR NFV DEPLOYMENTS

Red Hat OpenStack Platform (RHOSP) supports the following NFV deployments using director:

- Single root I/O virtualization (SR-IOV)
- Open vSwitch with Data Plane Development Kit (OVS-DPDK)

Additionally, you can deploy RHOSP with any of the following features:

- Composable roles
- Hyperconverged infrastructure guide
- Configuring Real-time Compute
- OVS hardware offload

RHOSP NFV deployments with Open Virtual Network (OVN) as the default Software Defined Networking (SDN) solution are unsupported. The following RHOSP NFV OVN configurations are supported:

- OVN with OVS-DPDK colocated with SR-IOV
- OVN with OVS TC Flower offload

#### 3.2.1. Deploying RHOSP with the OVS mechanism driver

Deploy RHOSP with the OVS mechanism driver:

**Procedure**

1. Modify the `containers-prepare-parameter.yaml` file so that the `neutron_driver` parameter is set to `ovs`.

```yaml
parameter_defaults:
  ContainerImagePrepare:
    - push_destination: true
```
2. Include the neutron-ovs.yaml environment file in the /usr/share/openstack-tripleo-heat-templates/environments/services directory with your deployment script.

```bash
TEMPLATE=$(/usr/share/openstack-tripleo-heat-templates

openstack overcloud deploy --templates \
-e ${{TEMPLATE}}/environments/network-environment.yaml \
-e ${{TEMPLATE}}/environments/network-isolation.yaml \
-e ${{TEMPLATE}}/environments/services/neutron-ovs.yaml \
-e ${{TEMPLATE}}/environments/services/neutron-ovs-dpdk.yaml \
-e ${{TEMPLATE}}/environments/services/neutron-sriov.yaml \
-e /home/stack/containers-prepare-parameter.yaml
```

### 3.2.2. Deploying OVN with OVS-DPDK and SR-IOV

Deploy DPDK and SRIOV VMs on the same node as OVN.

**Procedure**

1. Generate the **ComputeOvsDpdkSriov** role:

   ```bash
   openstack overcloud roles generate -o roles_data.yaml Controller ComputeOvsDpdkSriov
   ```

2. Add **OS::TripleO::Services::OVNMetadataAgent** to the Controller role.

3. Add the custom resources for OVS-DPDK with the **resource_registry** parameter:

   ```yaml
   resource_registry:
     # Specify the relative/absolute path to the config files you want to use for override the default.
     OS::TripleO::ComputeOvsDpdkSriov::Net::SoftwareConfig:
       nic-configs/computeovsdpdkxhrsriov.yaml
     OS::TripleO::Controller::Net::SoftwareConfig:
       nic-configs/controller.yaml
   ```

4. In the parameter_defaults section, edit the value of the tunnel type parameter to **geneve**:

   ```yaml
   NeutronTunnelTypes: 'geneve'
   NeutronNetworkType: ['geneve', 'vlan']
   ```

5. Optional: If you use a centralized routing model, disable Distributed Virtual Routing (DVR):

   ```yaml
   NeutronEnableDVR: false
   ```

6. Under **parameters_defaults**, set the bridge mapping:

   ```yaml
   # The OVS logical-to-physical bridge mappings to use.
   NeutronBridgeMappings: "datacentre:br-ex,data1:br-link0,data2:br-link1"
   ```
7. Configure the network interfaces in the `computeovsdpksriov.yaml` file:

```yaml
- type: ovs_user_bridge
  name: br-link0
  use_dhcp: false
  ovs_extra:
    - str_replace:
        template: set port br-link0 tag=_VLAN_TAG_
        params:
          _VLAN_TAG_: get_param: TenantNetworkVlanID
  addresses:
    - ip_netmask:
        get_param: TenantIpSubnet
  members:
    - type: ovs_dpdk_port
      name: br-link0-dpdk-port0
      rx_queue: 1
      members:
        - type: interface
          name: eno3
    - type: sriov_pf
      name: eno4
      use_dhcp: false
      numvfs: 5
      defroute: false
      nm_controlled: true
      hotplug: true
      promisc: false
```

8. Include the following yaml files in your deployment script:

   - neutron-ovn-dpdk.yaml
   - neutron-ovn-sriov.yaml

**NOTE**

Open Virtual Networking (OVN) is the default networking mechanism driver in Red Hat OpenStack Platform 17.1-Beta. If you want to use OVN with distributed virtual routing (DVR), you must include the `environments/services/neutron-ovn-dvr-ha.yaml` file in the `openstack overcloud deploy` command. If you want to use OVN without DVR, you must include the `environments/services/neutron-ovn-ha.yaml` file in the `openstack overcloud deploy` command, and set the `NeutronEnableDVR` parameter to `false`. If you want to use OVN with SR-IOV, you must include the `environments/services/neutron-ovn-sriov.yaml` file as the last of the OVN environment files in the `openstack overcloud deploy` command.

### 3.2.3. Deploying OVN with OVS TC Flower offload

Deploy OVS TC Flower offload on the same node as OVN.

**Procedure**

1. Generate the `ComputeOvsDpdkSriov` role:
openstack overcloud roles generate -o roles_data.yaml ControllerSriov ComputeSriov

2. Configure the physical_network parameter settings relevant to your deployment.

- For VLAN, set the physical_network parameter to the name of the network that you create in neutron after deployment. Use this value for the NeutronBridgeMappings parameter also.

- Under role-specific parameters, such as ComputeSriovOffloadParameters, ensure the value of the OvsHwOffload parameter is true.

```yaml
parameter_defaults:
  NeutronBridgeMappings: 'datacentre:br-ex,tenant:br-offload'
  NeutronNetworkVLANRanges: 'tenant:502:505'
  NeutronFlatNetworks: 'datacentre,tenant'
  NeutronPhysicalDevMappings:
    - tenant:ens1f0
    - tenant:ens1f1
  NovaPCIPassthrough:
    - address: "0000:17:00.1"
      physical_network: "tenant"
    - address: "0000:3b:00.1"
      physical_network: "tenant"
  NeutronTunnelTypes: "tenant"
  NeutronNetworkType: 'vlan'
  ComputeSriovOffloadParameters:
    OvsHwOffload: True
    KernelArgs: ",default_hugepagesz=1GB Hugepagesz=1G Hugepages=32"
    IsolCpusList: "1-11,13-23"
    NovaReservedHostMemory: 4096
    NovaComputeCpuDedicatedSet: ['1-11','13-23']
    NovaComputeCpuSharedSet: ['0','12']

3. Configure the network interfaces in the computeovsdpdksriov.yaml file:

```yaml
- type: ovs_bridge
  name: br-offload
  mtu: 9000
  use_dhcp: false
  addresses:
    - ip_netmask:
      get_param: TenantIpSubnet
  members:
    - type: linux_bond
      name: bond-pf
      bonding_options: "mode=active-backup miimon=100"
      members:
        - type: sriov_pf
          name: ens1f0
          numvfs: 3
          primary: true
          promisc: true
          use_dhcp: false
```
defroute: false
link_mode: switchdev
- type: sriov_pf
  name: ens1f1
  numvfs: 3
  promisc: true
  use_dhcp: false
  defroute: false
  link_mode: switchdev

4. Include the following yaml files in your deployment script:

- ovs-hw-offload.yaml
- neutron-ovn-sriov.yaml

```bash
TEMPLATES_HOME="/usr/share/openstack-tripleo-heat-templates"
CUSTOM_TEMPLATES="/home/stack/templates"

openstack overcloud deploy --templates \
  -r ${CUSTOM_TEMPLATES}/roles_data.yaml \
  -e ${TEMPLATES_HOME}/environments/services/neutron-ovn-sriov.yaml \
  -e ${TEMPLATES_HOME}/environments/ovs-hw-offload.yaml \
  -e ${CUSTOM_TEMPLATES}/network-environment.yaml
```

3.3. SUPPORTED DRIVERS

For a complete list of supported drivers, see Component, Plug-In, and Driver Support in Red Hat OpenStack Platform.

For a list of NICs tested for Red Hat OpenStack Platform deployments with NFV, see Tested NICs.

3.4. COMPATIBILITY WITH THIRD-PARTY SOFTWARE

For a complete list of products and services tested, supported, and certified to perform with Red Hat OpenStack Platform, see Third Party Software compatible with Red Hat OpenStack Platform. You can filter the list by product version and software category.

For a complete list of products and services tested, supported, and certified to perform with Red Hat Enterprise Linux, see Third Party Software compatible with Red Hat Enterprise Linux. You can filter the list by product version and software category.
CHAPTER 4. NETWORK CONSIDERATIONS

The undercloud host requires at least the following networks:

- **Provisioning network** - Provides DHCP and PXE-boot functions to help discover bare-metal systems for use in the overcloud.

- **External network** - A separate network for remote connectivity to all nodes. The interface connecting to this network requires a routable IP address, either defined statically, or generated dynamically from an external DHCP service.

The minimal overcloud network configuration includes the following NIC configurations:

- **Single NIC configuration** - One NIC for the provisioning network on the native VLAN and tagged VLANs that use subnets for the different overcloud network types.

- **Dual NIC configuration** - One NIC for the provisioning network and the other NIC for the external network.

- **Dual NIC configuration** - One NIC for the provisioning network on the native VLAN, and the other NIC for tagged VLANs that use subnets for different overcloud network types.

- **Multiple NIC configuration** - Each NIC uses a subnet for a different overcloud network type.

For more information on the networking requirements, see Networking requirements.
CHAPTER 5. PLANNING AN SR-IOV DEPLOYMENT

Optimize single root I/O virtualization (SR-IOV) deployments for NFV by setting individual parameters based on your Compute node hardware.

See Discovering your NUMA node topology to evaluate your hardware impact on the SR-IOV parameters.

5.1. HARDWARE PARTITIONING FOR AN SR-IOV DEPLOYMENT

To achieve high performance with SR-IOV, partition the resources between the host and the guest.

A typical topology includes 14 cores per NUMA node on dual socket Compute nodes. Both hyper-threading (HT) and non-HT cores are supported. Each core has two sibling threads. One core is dedicated to the host on each NUMA node. The virtual network function (VNF) handles the SR-IOV interface bonding. All the interrupt requests (IRQs) are routed on the host cores. The VNF cores are dedicated to the VNFs. They provide isolation from other VNFs and isolation from the host. Each VNF must use resources on a single NUMA node. The SR-IOV NICs used by the VNF must also be associated with that same NUMA node. This topology does not have a virtualization overhead. The host, OpenStack Networking (neutron), and Compute (nova) configuration parameters are exposed in a single file for ease, consistency, and to avoid incoherence that is fatal to proper isolation, causing preemption, and packet loss. The host and virtual machine isolation depend on a tuned profile, which defines the boot parameters and any Red Hat OpenStack Platform modifications based on the list of isolated CPUs.

5.2. TOPOLOGY OF AN NFV SR-IOV DEPLOYMENT

The following image has two VNFs each with the management interface represented by mgt and the data plane interfaces. The management interface manages the ssh access, and so on. The data plane interfaces bond the VNFs to DPDK to ensure high availability, as VNFs bond the data plane interfaces using the DPDK library. The image also has two provider networks for redundancy. The Compute node has two regular NICs bonded together and shared between the VNF management and the Red Hat OpenStack Platform API management.
The image shows a VNF that uses DPDK at an application level, and has access to SR-IOV virtual functions (VFs) and physical functions (PFs), for better availability or performance, depending on the fabric configuration. DPDK improves performance, while the VF/PF DPDK bonds provide support for failover, and high availability. The VNF vendor must ensure that the DPDK poll mode driver (PMD) supports the SR-IOV card that is being exposed as a VF/PF. The management network uses OVS, therefore the VNF sees a mgmt network device using the standard virtIO drivers. You can use that device to initially connect to the VNF, and ensure that the DPDK application bonds the two VF/PFs.

5.2.1. Topology for NFV SR-IOV without HCI

Observe the topology for SR-IOV without hyper-converged infrastructure (HCI) for NFV in the image below. It consists of compute and controller nodes with 1 Gbps NICs, and the director node.
CHAPTER 6. DEPLOYING SR-IOV TECHNOLOGIES

In your Red Hat OpenStack Platform NFV deployment, you can achieve higher performance with single root I/O virtualization (SR-IOV), when you configure direct access from your instances to a shared PCIe resource through virtual resources.

6.1. PREREQUISITES FOR DEPLOYING SR-IOV TECHNOLOGIES

- For details on how to install and configure the undercloud before deploying the overcloud, see the Installing and managing Red Hat OpenStack Platform with director Guide.

**NOTE**

Do not manually edit any values in /etc/tuned/cpu-partitioning-variables.conf that director heat templates modify.

6.2. CONFIGURING SR-IOV

To deploy Red Hat OpenStack Platform (RHOSP) with single root I/O virtualization (SR-IOV), configure the shared PCIe resources that have SR-IOV capabilities that instances can request direct access to.

**NOTE**

The following CPU assignments, memory allocation, and NIC configurations are examples, and might be different from your use case.

**Procedure**

1. Log in to the undercloud as the stack user.
2. Source the stackrc file:

   ```
   [stack@director ~]$ source ~/stackrc
   ```
3. Generate a new roles data file named roles_data_compute_sriov.yaml that includes the Controller and ComputeSriov roles:

   ```
   (undercloud)$ openstack overcloud roles \
   generate -o /home/stack/templates/roles_data_compute_sriov.yaml \
   Controller ComputeSriov
   ```

   **ComputeSriov** is a custom role provided with your RHOSP installation that includes the NeutronSriovAgent, NeutronSriovHostConfig services, in addition to the default compute services.
4. To prepare the SR-IOV containers, include the neutron-sriov.yaml and roles_data_compute_sriov.yaml files when you generate the overcloud_images.yaml file.

   ```
   $ sudo openstack tripleo container image prepare \
   --roles-file ~/templates/roles_data_compute_sriov.yaml \
   -e /usr/share/openstack-tripleo-heat-templates/environments/services/neutron-sriov.yaml \
   -e ~/containers-prepare-parameter.yaml \ 
   --output-env-file=/home/stack/templates/overcloud_images.yaml
   ```
For more information on container image preparation, see Preparing container images in the Installing and managing Red Hat OpenStack Platform with director guide.

5. Create a copy of the `/usr/share/openstack-tripleo-heat-templates/environments/network-environment.yaml` file in your environment file directory:

   ```
   ```

6. Add the following parameters under `parameter_defaults` in your `network-environment-sriov.yaml` file to configure the SR-IOV nodes for your cluster and your hardware configuration:

   ```
   NeutronNetworkType: 'vlan'
   NeutronNetworkVLANRanges:
     - tenant:22:22
     - tenant:25:25
   NeutronTunnelTypes: "
   ```

7. To determine the `vendor_id` and `product_id` for each PCI device type, use one of the following commands on the physical server that has the PCI cards:

   - To return the `vendor_id` and `product_id` from a deployed overcloud, use the following command:
     ```
     # lspci -nn -s <pci_device_address>
     3b:00.0 Ethernet controller [0200]: Intel Corporation Ethernet Controller X710 for 10GbE SFP+ [<vendor_id>: <product_id>] (rev 02)
     ```
   - To return the `vendor_id` and `product_id` of a physical function (PF) if you have not yet deployed the overcloud, use the following command:
     ```
     (undercloud) [stack@undercloud-0 ~]$ openstack baremetal introspection data save <baremetal_node_name> | jq ".inventory.interfaces[] | .name, .vendor, .product"
     ```

8. Configure role specific parameters for SR-IOV compute nodes in your `network-environment-sriov.yaml` file:

   ```
   ComputeSriovParameters:
     IsolCpusList: "1-19,21-39"
     KernelArgs: "default_hugepagesz=1GB hugepagesz=1G hugepages=32 iommu=pt intel_iommu=on isolcpus=1-19,21-39"
     TunedProfileName: "cpu-partitioning"
   NeutronBridgeMappings:
     - tenant:br-link0
   NeutronPhysicalDevMappings:
     - tenant:p7p1
   NovaComputeCpuDedicatedSet: '1-19,21-39'
   NovaReservedHostMemory: 4096
   ```

**NOTE**

The `NovaVcpuPinSet` parameter is now deprecated, and is replaced by `NovaComputeCpuDedicatedSet` for dedicated, pinned workloads.
9. Configure the PCI passthrough devices for the SR-IOV compute nodes in your network-environment-sriov.yaml file:

```yaml
ComputeSriovParameters:
  ...
  NovaPCIPassthrough:
    - vendor_id: "<vendor_id>"
      product_id: "<product_id>"
      address: <NIC_address>
      physical_network: "<physical_network>"
    ...
```

- Replace `<vendor_id>` with the vendor ID of the PCI device.
- Replace `<product_id>` with the product ID of the PCI device.
- Replace `<NIC_address>` with the address of the PCI device. For information about how to configure the `address` parameter, see Guidelines for configuring NovaPCIPassthrough in the Configuring the Compute Service for Instance Creation guide.
- Replace `<physical_network>` with the name of the physical network the PCI device is located on.

**NOTE**

Do not use the `devname` parameter when you configure PCI passthrough because the device name of a NIC can change. To create a Networking service (neutron) port on a PF, specify the `vendor_id`, the `product_id`, and the PCI device address in NovaPCIPassthrough, and create the port with the `--vnic-type direct-physical` option. To create a Networking service port on a virtual function (VF), specify the `vendor_id` and `product_id` in NovaPCIPassthrough, and create the port with the `--vnic-type direct` option. The values of the `vendor_id` and `product_id` parameters might be different between physical function (PF) and VF contexts. For more information about how to configure NovaPCIPassthrough, see Guidelines for configuring NovaPCIPassthrough in the Configuring the Compute Service for Instance Creation guide.

10. Configure the SR-IOV enabled interfaces in the compute.yaml network configuration template.

To create SR-IOV VFs, configure the interfaces as standalone NICs:

```yaml
- type: sriov_pf
  name: p7p3
  mtu: 9000
  numvfs: 10
  use_dhcp: false
  defroute: false
  nm_controlled: true
  hotplug: true
  promisc: false

- type: sriov_pf
  name: p7p4
  mtu: 9000
  numvfs: 10
```
NOTE

The numvfs parameter replaces the NeutronSriovNumVFs parameter in the network configuration templates. Red Hat does not support modification of the NeutronSriovNumVFs parameter or the numvfs parameter after deployment. If you modify either parameter after deployment, it might cause a disruption for the running instances that have an SR-IOV port on that PF. In this case, you must hard reboot these instances to make the SR-IOV PCI device available again.

11. Ensure that the list of default filters includes the value AggregateInstanceExtraSpecsFilter:

```
parameter_defaults:
  NovaSchedulerEnabledFilters:
    - AvailabilityZoneFilter
    - ComputeFilter
    - ComputeCapabilitiesFilter
    - ImagePropertiesFilter
    - ServerGroupAntiAffinityFilter
    - ServerGroupAffinityFilter
    - PciPassthroughFilter
    - AggregateInstanceExtraSpecsFilter
```

12. Run the `overcloud_deploy.sh` script.

### 6.3. CONFIGURING NIC PARTITIONING

You can reduce the number of NICs that you need for each host by configuring single root I/O virtualization (SR-IOV) virtual functions (VFs) for Red Hat OpenStack Platform (RHOSP) management networks and provider networks. When you partition a single, high-speed NIC into multiple VFs, you can use the NIC for both control and data plane traffic. This feature has been validated on Intel Fortville NICs, and Mellanox CX-5 NICs.

#### Procedure

1. Open the NIC config file for your chosen role.

2. Add an entry for the interface type `sriov_pf` to configure a physical function that the host can use:

```
- type: sriov_pf
  name: <interface_name>
  use_dhcp: false
  numvfs: <number_of_vfs>
  promisc: <true/false>
```

- Replace `<interface_name>` with the name of the interface.
- Replace `<number_of_vfs>` with the number of VFs.
• Optional: Replace `<true/false>` with `true` to set promiscuous mode, or `false` to disable promiscuous mode. The default value is `true`.

NOTE

The `numvfs` parameter replaces the `NeutronSriovNumVFs` parameter in the network configuration templates. Red Hat does not support modification of the `NeutronSriovNumVFs` parameter or the `numvfs` parameter after deployment. If you modify either parameter after deployment, it might cause a disruption for the running instances that have an SR-IOV port on that physical function (PF). In this case, you must hard reboot these instances to make the SR-IOV PCI device available again.

3. Add an entry for the interface type `sriov_vf` to configure virtual functions that the host can use:

```yaml
- type: <bond_type>
  name: internal_bond
  bonding_options: mode=<bonding_option>
  use_dhcp: false
  members:
    - type: sriov_vf
      device: <pf_device_name>
      vfid: <vf_id>
    - type: sriov_vf
      device: <pf_device_name>
      vfid: <vf_id>
  - type: vlan
    vlan_id:
      get_param: InternalApiNetworkVlanID
      spoofcheck: false
      device: internal_bond
    addresses:
      - ip_netmask:
          get_param: InternalApiIpSubnet
          routes:
            list_concat_unique:
              - get_param: InternalApiInterfaceRoutes

- type: vlan
  vlan_id:
    get_param: InternalApiNetworkVlanID
    spoofcheck: false
    device: internal_bond
  addresses:
    - ip_netmask:
        get_param: InternalApiIpSubnet
        routes:
          list_concat_unique:
            - get_param: InternalApiInterfaceRoutes
```

• Replace `<bond_type>` with the required bond type, for example, `linux_bond`. You can apply VLAN tags on the bond for other bonds, such as `ovs_bond`.

• Replace `<bonding_option>` with one of the following supported bond modes:
  
  - active-backup
  - Balance-slb

NOTE

LACP bonds are not supported.

• Specify the `sriov_vf` as the interface type to bond in the `members` section.
NOTE

If you are using an OVS bridge as the interface type, you can configure only one OVS bridge on the `sriov_vf` of a `sriov_pf` device. More than one OVS bridge on a single `sriov_pf` device can result in packet duplication across VFs, and decreased performance.

- Replace `<pf_device_name>` with the name of the PF device.
- If you use a `linux_bond`, you must assign VLAN tags. If you set a VLAN tag, ensure that you set a unique tag for each VF associated with a single `sriov_pf` device. You cannot have two VFs from the same PF on the same VLAN.
- Replace `<vf_id>` with the ID of the VF. The applicable VF ID range starts at zero, and ends at the maximum number of VFs minus one.
- Disable spoof checking.
- Apply VLAN tags on the `sriov_vf` for `linux_bond` over VFs.

4. To reserve VFs for instances, include the `NovaPCIPassthrough` parameter in an environment file, for example:

   ```
   NovaPCIPassthrough:
   - address: "0000:19:0e.3"
     trusted: "true"
     physical_network: "sriov1"
   - address: "0000:19:0e.0"
     trusted: "true"
     physical_network: "sriov2"
   
   Director identifies the host VFs, and derives the PCI addresses of the VFs that are available to the instance.
   
   5. Enable `IOMMU` on all nodes that require NIC partitioning. For example, if you want NIC Partitioning for Compute nodes, enable IOMMU using the `KernelArgs` parameter for that role:

   ```
   parameter_defaults:
   ComputeParameters:
     KernelArgs: "intel_iommu=on iommu=pt"
   ```

   NOTE

   When you first add the `KernelArgs` parameter to the configuration of a role, the overcloud nodes are automatically rebooted. If required, you can disable the automatic rebooting of nodes and instead perform node reboots manually after each overcloud deployment. For more information, see Configuring manual node reboot to define `KernelArgs`.

6. Add your role file and environment files to the stack with your other environment files and deploy the overcloud:

   ```
   (undercloud)$ openstack overcloud deploy --templates \
   -r os-net-config.yaml \
   -e [your environment files] \
   ```
-e /home/stack/templates/<compute_environment_file>.yaml

Validation

1. Log in to the overcloud Compute node as **tripleo-admin** and check the number of VFs:

   ```bash
   [tripleo-admin@overcloud-compute-0 tripleo-admin]$ sudo cat /sys/class/net/p4p1/device/sriov_numvfs
   10
   [tripleo-admin@overcloud-compute-0 tripleo-admin]$ sudo cat /sys/class/net/p4p2/device/sriov_numvfs
   10
   ```

2. Show OVS connections:

   ```bash
   [tripleo-admin@overcloud-compute-0]$ sudo ovs-vsctl show
   b6567fa8-c9ec-4247-9a08-cbf34f04c85f
   Manager "ptcp:6640:127.0.0.1"
      is_connected: true
   Bridge br-sriov2
      Controller "tcp:127.0.0.1:6633"
         is_connected: true
         fail_mode: secure
         datapath_type: netdev
      Port phy-br-sriov2
         Interface phy-br-sriov2
            type: patch
            options: {peer=int-br-sriov2}
      Port br-sriov2
         Interface br-sriov2
            type: internal
   Bridge br-sriov1
      Controller "tcp:127.0.0.1:6633"
         is_connected: true
         fail_mode: secure
         datapath_type: netdev
      Port phy-br-sriov1
         Interface phy-br-sriov1
            type: patch
            options: {peer=int-br-sriov1}
      Port br-sriov1
         Interface br-sriov1
            type: internal
   Bridge br-ex
      Controller "tcp:127.0.0.1:6633"
         is_connected: true
         fail_mode: secure
         datapath_type: netdev
      Port br-ex
         Interface br-ex
            type: internal
      Port phy-br-ex
         Interface phy-br-ex
            type: patch
            options: {peer=int-br-ex}
   ```
Bridge br-tenant
Controller "tcp:127.0.0.1:6633"
  is_connected: true
  fail_mode: secure
data_path_type: netdev
Port br-tenant
tag: 305
  Interface br-tenant
type: internal
Port phy-br-tenant
  Interface phy-br-tenant
type: patch
  options: {peer=int-br-tenant}
Port dpdkbond0
  Interface dpdk0
type: dpdk
  options: {dpdk-devargs="0000:18:0e.0"}
  Interface dpdk1
type: dpdk
  options: {dpdk-devargs="0000:18:0a.0"}
Bridge br-tun
Controller "tcp:127.0.0.1:6633"
  is_connected: true
  fail_mode: secure
data_path_type: netdev
Port vxlan-98140025
  Interface vxlan-98140025
type: vxlan
  options: {df_default="true", egress_pkt_mark="0", in_key=flow,
local_ip="152.20.0.229", out_key=flow, remote_ip="152.20.0.37"}
Port br-tun
  Interface br-tun
type: internal
Port patch-int
  Interface patch-int
type: patch
  options: {peer=patch-tun}
Port vxlan-98140015
  Interface vxlan-98140015
type: vxlan
  options: {df_default="true", egress_pkt_mark="0", in_key=flow,
local_ip="152.20.0.229", out_key=flow, remote_ip="152.20.0.21"}
Port vxlan-9814009f
  Interface vxlan-9814009f
type: vxlan
  options: {df_default="true", egress_pkt_mark="0", in_key=flow,
local_ip="152.20.0.229", out_key=flow, remote_ip="152.20.0.159"}
Port vxlan-981400cc
  Interface vxlan-981400cc
type: vxlan
  options: {df_default="true", egress_pkt_mark="0", in_key=flow,
local_ip="152.20.0.229", out_key=flow, remote_ip="152.20.0.204"}
Bridge br-int
Controller "tcp:127.0.0.1:6633"
  is_connected: true
  fail_mode: secure
3. Log in to your OVS-DPDK SR-IOV Compute node as `tripleo-admin` and check Linux bonds:

```
[tripleo-admin@overcloud-computeovsdpdksriov-1 ~]$ cat /proc/net/bonding/<bond_name>
```

Ethernet Channel Bonding Driver: v3.7.1 (April 27, 2011)

Bonding Mode: fault-tolerance (active-backup)
Primary Slave: None
Currently Active Slave: eno3v1
MII Status: up
MII Polling Interval (ms): 0
Up Delay (ms): 0
Down Delay (ms): 0
Peer Notification Delay (ms): 0

Slave Interface: eno3v1
MII Status: up
Speed: 10000 Mbps
Duplex: full
Link Failure Count: 0
Permanent HW addr: 4e:77:94:bd:38:d2
Slave queue ID: 0

Slave Interface: eno4v1
MII Status: up
Speed: 10000 Mbps
Duplex: full
Link Failure Count: 0
 Permanent HW addr: 4a:74:52:a7:aa:7c
 Slave queue ID: 0

4. List OVS bonds:

```
[tripleo-admin@overcloud-computeovsdpdksriov-1 ~]$ sudo ovs-appctl bond/show
---- dpdkbond0 ----
bond_mode: balance-slb
bond may use recirculation: no, Recirc-ID : -1
bond-hash-basis: 0
updelay: 0 ms
downdelay: 0 ms
next rebalance: 9491 ms
lacp_status: off
lacp_fallback_ab: false
active slave mac: ce:ee:c7:58:8e:b2(dpdk1)

slave dpdk0: enabled
may_enable: true

slave dpdk1: enabled
active slave
may_enable: true
```

If you used `NovaPCIPassthrough` to pass VFs to instances, test by deploying an SR-IOV instance.

### 6.4. EXAMPLE NIC PARTITIONING CONFIGURATIONS

**Linux bond over VFs**

The following example configures a Linux bond over VFs, disables `spoofcheck`, and applies VLAN tags to `sriov_vf`:

```
- type: linux_bond
  name: bond_api
  bonding_options: "mode=active-backup"
  members:
    - type: sriov_vf
      device: eno2
      vfid: 1
      vlan_id:
        get_param: InternalApiNetworkVlanID
      spoofcheck: false
    - type: sriov_vf
      device: eno3
      vfid: 1
      vlan_id:
        get_param: InternalApiNetworkVlanID
      spoofcheck: false
  addresses:
    - ip_netmask:
        get_param: InternalApiIpSubnet
```
routes:
  list_concat_unique:
  - get_param: InternalApiInterfaceRoutes

OVS bridge on VFs

The following example configures an OVS bridge on VFs:

- type: ovs_bridge
  name: br-bond
  use_dhcp: true
  members:
  - type: vlan
    vlan_id:
      get_param: TenantNetworkVlanID
  addresses:
  - ip_netmask:
    get_param: TenantIpSubnet
  routes:
    list_concat_unique:
    - get_param: ControlPlaneStaticRoutes
  - type: ovs_bond
    name: bond_vf
    ovs_options: "bond_mode=active-backup"
    members:
    - type: sriov_vf
      device: p2p1
      vfid: 2
    - type: sriov_vf
      device: p2p2
      vfid: 2

OVS user bridge on VFs

The following example configures an OVS user bridge on VFs and applies VLAN tags to ovs_user_bridge:

- type: ovs_user_bridge
  name: br-link0
  use_dhcp: false
  mtu: 9000
  ovs_extra:
  - str_replace:
    template: set port br-link0 tag=_VLAN_TAG_
    params:
      _VLAN_TAG_:
      get_param: TenantNetworkVlanID
  addresses:
  - ip_netmask:
    list_concat_unique:
    - get_param: TenantInterfaceRoutes
  members:
  - type: ovs_dpdk_bond
    name: dpdkbond0
    mtu: 9000
    ovs_extra:
- set port dpdkbond0 bond_mode=balance-slb
members:
- type: ovs_dpdk_port
  name: dpdk0
members:
- type: sriov_vf
device: eno2
vfid: 3
- type: ovs_dpdk_port
  name: dpdk1
members:
- type: sriov_vf
device: eno3
vfid: 3

6.5. CONFIGURING OVS HARDWARE OFFLOAD

The procedure for OVS hardware offload configuration shares many of the same steps as configuring SR-IOV.

Procedure

1. Generate an overcloud role for OVS hardware offload that is based on the Compute role:

   ```bash
   openstack overcloud roles generate -o roles_data.yaml Controller
   Compute:ComputeOvsHwOffload
   ```

2. Optional: Change the `HostnameFormatDefault: '%stackname%-compute-%index%` name for the `ComputeOvsHwOffload` role.

3. Add the `OvsHwOffload` parameter under role-specific parameters with a value of `true`.

4. To configure neutron to use the iptables/hybrid firewall driver implementation, include the line:
   `NeutronOVSFirewallDriver: iptables_hybrid`. For more information about `NeutronOVSFirewallDriver`, see Using the Open vSwitch Firewall in the Hardening Red Hat OpenStack Platform Guide.

5. Configure the `physical_network` parameter to match your environment.
   - For VLAN, set the `physical_network` parameter to the name of the network you create in neutron after deployment. This value should also be in `NeutronBridgeMappings`.
   - For VXLAN, set the `physical_network` parameter to `null`.
     Example:

     ```yaml
     parameter_defaults:
      NeutronOVSFirewallDriver: iptables_hybrid
      ComputeSriovParameters:
       IsolCpusList: 2-9,21-29,11-19,31-39
       KernelArgs: "default_hugepagesz=1GB hugepagesz=1G hugepages=128 intel_iommu=on iommu=pt"
      OvsHwOffload: true
      TunedProfileName: "cpu-partitioning"
      NeutronBridgeMappings:
       - tenant:br-tenant
     ```
NovaPCIPassthrough:
- vendor_id: <vendor-id>
  product_id: <product-id>
  address: <address>
  physical_network: "tenant"
- vendor_id: <vendor-id>
  product_id: <product-id>
  address: <address>
  physical_network: "null"
NovaReservedHostMemory: 4096
NovaComputeCpuDedicatedSet: 1-9,21-29,11-19,31-39

- Replace `<vendor-id>` with the vendor ID of the physical NIC.
- Replace `<product-id>` with the product ID of the NIC VF.
- Replace `<address>` with the address of the physical NIC.

For more information about how to configure NovaPCIPassthrough, see Guidelines for configuring NovaPCIPassthrough in the Configuring the Compute Service for Instance Creation guide.

6. Ensure that the list of default filters includes NUMATopologyFilter:

```
parameter_defaults:
NovaSchedulerEnabledFilters:
  - AvailabilityZoneFilter
  - ComputeFilter
  - ComputeCapabilitiesFilter
  - ImagePropertiesFilter
  - ServerGroupAntiAffinityFilter
  - ServerGroupAffinityFilter
  - PciPassthroughFilter
  - NUMATopologyFilter
```

7. Configure one or more network interfaces intended for hardware offload in the `compute-sriov.yaml` configuration file:

```
- type: ovs_bridge
  name: br-tenant
  mtu: 9000
  members:
    - type: sriov_pf
      name: p7p1
      numvfs: 5
      mtu: 9000
      primary: true
      promisc: true
      use_dhcp: false
      link_mode: switchdev
```

NOTE
Optional: For details on how to troubleshoot and configure OVS Hardware Offload issues in RHOSP 17.1-Beta with Mellanox ConnectX5 NICs, see Troubleshooting Hardware Offload.
NOTE

- Do not use the `NeutronSriovNumVFs` parameter when configuring Open vSwitch hardware offload. The number of virtual functions is specified using the `numvfs` parameter in a network configuration file used by `os-net-config`. Red Hat does not support modifying the `numvfs` setting during update or redeployment.

- Do not configure Mellanox network interfaces as a nic-config interface type `ovs-vlan` because this prevents tunnel endpoints such as VXLAN from passing traffic due to driver limitations.

8. Include the `ovs-hw-offload.yaml` file in the `overcloud deploy` command:

   ```bash
   TEMPLATES_HOME=/usr/share/openstack-tripleo-heat-templates
   CUSTOM_TEMPLATES=/home/stack/templates

   openstack overcloud deploy --templates "
   -r ${CUSTOM_TEMPLATES}/roles_data.yaml
   -e ${TEMPLATES_HOME}/environments/ovs-hw-offload.yaml
   -e ${CUSTOM_TEMPLATES}/network-environment.yaml
   -e ${CUSTOM_TEMPLATES}/neutron-ovs.yaml"
   ```

6.5.1. Verifying OVS hardware offload

1. Confirm that a PCI device is in `switchdev` mode:

   ```bash
   # devlink dev eswitch show pci/0000:03:00.0
   pci/0000:03:00.0: mode switchdev inline-mode none encap enable
   ```

2. Verify if offload is enabled in OVS:

   ```bash
   # ovs-vsctl get Open_vSwitch . other_config:hw-offload
   "true"
   ```

6.6. TUNING EXAMPLES FOR OVS HARDWARE OFFLOAD

For optimal performance you must complete additional configuration steps.

Adjusting the number of channels for each network interface to improve performance

A channel includes an interrupt request (IRQ) and the set of queues that trigger the IRQ. When you set the `mlx5_core` driver to `switchdev` mode, the `mlx5_core` driver defaults to one combined channel, which might not deliver optimal performance.

Procedure

- On the PF representors, enter the following command to adjust the number of CPUs available to the host. Replace `$(nproc)` with the number of CPUs you want to make available:

  ```bash
  $ sudo ethtool -L enp3s0f0 combined $(nproc)
  ```
CPU pinning

To prevent performance degradation from cross-NUMA operations, locate NICs, their applications, the VF guest, and OVS in the same NUMA node. For more information, see Configuring CPU pinning on Compute nodes in the Configuring the Compute Service for Instance Creation guide.

6.7. COMPONENTS OF OVS HARDWARE OFFLOAD

A reference for configuring and troubleshooting the components of OVS HW Offload with Mellanox smart NICs.

Nova

Configure the Nova scheduler to use the NovaPCIPassthrough filter with the NUMATopologyFilter and DerivePciWhitelistEnabled parameters. When you enable OVS HW Offload, the Nova scheduler operates similarly to SR-IOV passthrough for instance spawning.

Neutron

When you enable OVS HW Offload, use the devlink cli tool to set the NIC e-switch mode to switchdev. Switchdev mode establishes representor ports on the NIC that are mapped to the VFs.

Procedure

1. To allocate a port from a switchdev-enabled NIC, log in as an admin user, create a neutron port with a binding-profile value of capabilities, and disable port security:

   $ openstack port create --network private --vnic-type=direct --binding-profile '{"capabilities": ["switchdev"]}' direct_port1 --disable-port-security

   IMPORTANT
   You must enable security groups and port security on switchdev ports for the connection tracking (conntrack) module to offload openflow flows to hardware.

2. Pass this port information when you create the instance.
   You associate the representor port with the instance VF interface and connect the representor port to OVS bridge br-int for one-time OVS data path processing. A VF port representor functions like a software version of a physical “patch panel” front-end.

   For more information about new instance creation, see Deploying an SR-IOV instance.

OVS

In an environment with hardware offload configured, the first packet transmitted traverses the OVS kernel path, and this packet journey establishes the ml2 OVS rules for incoming and outgoing traffic for the instance traffic. When the flows of the traffic stream are established, OVS uses the traffic control (TC) Flower utility to push these flows on the NIC hardware.

Procedure

1. Use director to apply the following configuration on OVS:

   $ sudo ovs-vsctl set Open_vSwitch . other_config:hw-offload=true
2. Restart to enable HW Offload.

**Traffic Control (TC) subsystems**

When you enable the `hw-offload` flag, OVS uses the TC data path. TC Flower is an iproute2 utility that writes data path flows on hardware. This ensures that the flow is programmed on both the hardware and software data paths, for redundancy.

**Procedure**

1. Apply the following configuration. This is the default option if you do not explicitly configure `tc-policy`:
   
   ```
   $ sudo ovs-vsctl set Open_vSwitch . other_config:tc-policy=none
   ```

2. Restart OVS.

**NIC PF and VF drivers**

Mlx5_core is the PF and VF driver for the Mellanox ConnectX-5 NIC. The mlx5_core driver performs the following tasks:

- Creates routing tables on hardware.
- Manages network flow management.
- Configures the Ethernet switch device driver model, `switchdev`.
- Creates block devices.

**Procedure**

- Use the following `devlink` commands to query the mode of the PCI device.
  
  ```
  $ sudo devlink dev eswitch set pci/0000:03:00.0 mode switchdev
  $ sudo devlink dev eswitch show pci/0000:03:00.0
  pci/0000:03:00.0: mode switchdev inline-mode none encap enable
  ```

**NIC firmware**

The NIC firmware performs the following tasks:

- Maintains routing tables and rules.
- Fixes the pipelines of the tables.
- Manages hardware resources.
- Creates VFs.

The firmware works with the driver for optimal performance.
Although the NIC firmware is non-volatile and persists after you reboot, you can modify the configuration during run time.

**Procedure**

- Apply the following configuration on the interfaces, and the representor ports, to ensure that TC Flower pushes the flow programming at the port level:

  ```
  $ sudo ethtool -K enp3s0f0 hw-tc-offload on
  ```

**NOTE**

Ensure that you keep the firmware updated. **Yum** or **dnf** updates might not complete the firmware update. For more information, see your vendor documentation.

### 6.8. TROUBLESHOOTING OVS HARDWARE OFFLOAD

**Prerequisites**

- Linux Kernel 4.13 or newer
- OVS 2.8 or newer
- RHOSP 12 or newer
- Iproute 4.12 or newer
- Mellanox NIC firmware, for example FW ConnectX-5 16.21.0338 or newer

For more information about supported prerequisites, see see the Red Hat Knowledgebase solution Network Adapter Fast Datapath Feature Support Matrix.

**Configuring the network in an OVS HW offload deployment**

In a HW offload deployment, you can choose one of the following scenarios for your network configuration according to your requirements:

- You can base guest VMs on VXLAN and VLAN by using either the same set of interfaces attached to a bond, or a different set of NICs for each type.

- You can bond two ports of a Mellanox NIC by using Linux bond.

- You can host tenant VXLAN networks on VLAN interfaces on top of a Mellanox Linux bond.

Ensure that individual NICs and bonds are members of an ovs-bridge.

Refer to the below example network configuration:

```yaml
- type: ovs_bridge
  name: br-offload
  mtu: 9000
  use_dhcp: false
  members:
  - type: linux_bond
    name: bond-pf
```
bonding_options: "mode=active-backup miimon=100"
members:
- type: sriov_pf
  name: p5p1
  numvfs: 3
  primary: true
  promisc: true
  use_dhcp: false
  defroute: false
  link_mode: switchdev
- type: sriov_pf
  name: p5p2
  numvfs: 3
  promisc: true
  use_dhcp: false
  defroute: false
  link_mode: switchdev
- type: vlan
  vlan_id:
    get_param: TenantNetworkVlanID
  device: bond-pf
  addresses:
  - ip_netmask:
      get_param: TenantIpSubnet

The following bonding configurations are supported:

- active-backup - mode=1
- active-active or balance-xor - mode=2
- 802.3ad (LACP) - mode=4

The following bonding configuration is not supported:

- xmit_hash_policy=layer3+4

Verifying the interface configuration

Verify the interface configuration with the following procedure.

Procedure

1. During deployment, use the host network configuration tool `os-net-config` to enable `hw-tc-offload`.

2. Enable `hw-tc-offload` on the `sriov_config` service any time you reboot the Compute node.

3. Set the `hw-tc-offload` parameter to `on` for the NICs that are attached to the bond:

```
[root@overcloud-computesriov-0 ~]# ethtool -k ens1f0 | grep tc-offload
hw-tc-offload: on
```

Verifying the interface mode

Verify the interface mode with the following procedure.

42
Procedure

1. Set the eswitch mode to `switchdev` for the interfaces you use for HW offload.

2. Use the host network configuration tool `os-net-config` to enable `eswitch` during deployment.

3. Enable `eswitch` on the `sriov_config` service any time you reboot the Compute node.

```
[root@overcloud-computesriov-0 ~]# devlink dev eswitch show pci/$(ethtool -i ens1f0 | grep bus-info | cut -d ':' -f 2,3,4 | awk '{$1=$1};1')
```

**NOTE**

The driver of the PF interface is set to "mlx5e_rep", to show that it is a representor of the e-switch uplink port. This does not affect the functionality.

Verifying the offload state in OVS

Verify the offload state in OVS with the following procedure.

- Enable hardware offload in OVS in the Compute node.

```
[root@overcloud-computesriov-0 ~]# ovs-vsctl get Open_vSwitch . other_config:hw-offload "true"
```

Verifying the name of the VF representor port

To ensure consistent naming of VF representor ports, `os-net-config` uses udev rules to rename the ports in the `<PF-name>_<VF_id>` format.

**Procedure**

- After deployment, verify that the VF representor ports are named correctly.

```
root@overcloud-computesriov-0 ~]# cat /etc/udev/rules.d/80-persistent-os-net-config.rules
# This file is autogenerated by os-net-config

SUBSYSTEM=="net", ACTION=="add", ATTR{phys_switch_id}"=", ATTR{phys_port_name}"="pf\*vf\*", ENV{NM_UNMANAGED}"="1"
SUBSYSTEM=="net", ACTION=="add", DRIVERS=="\".*\", KERNELS=="0000:65:00.0", NAME="ens1f0"

SUBSYSTEM=="net", ACTION=="add", ATTR{phys_switch_id}"="98039b7f9e48", ATTR{phys_port_name}"="pf0vf\*", IMPORT{program}"="/etc/udev/rep-link-name.sh $attr{phys_port_name}", NAME="ens1f0\$_env{NUMBER}"

SUBSYSTEM=="net", ACTION=="add", ATTR{phys_switch_id}"="98039b7f9e49", ATTR{phys_port_name}"="pf1vf\*", IMPORT{program}"="/etc/udev/rep-link-name.sh $attr{phys_port_name}", NAME="ens1f1"

SUBSYSTEM=="net", ACTION=="add", ATTR{phys_switch_id}"="98039b7f9e48", ATTR{phys_port_name}"="pf0vf\*", IMPORT{program}"="/etc/udev/rep-link-name.sh $attr{phys_port_name}", NAME="ens1f0\$_env{NUMBER}"
```

Examining network traffic flow

HW offloaded network flow functions in a similar way to physical switches or routers with application-specific integrated circuit (ASIC) chips. You can access the ASIC shell of a switch or router to examine the routing table and for other debugging. The following procedure uses a Broadcom chipset from a
Cumulus Linux switch as an example. Replace the values that are appropriate to your environment.

Procedure

1. To get Broadcom chip table content, use the `bcmcmd` command.

   ```bash
   root@dni-7448-26:~# cl-bcmcmd l2 show
   mac=00:02:00:00:00:08 vlan=2000 GPORT=0x2 modid=0 port=2/xe1
   mac=00:02:00:00:00:09 vlan=2000 GPORT=0x2 modid=0 port=2/xe1 Hit
   ```

2. Inspect the Traffic Control (TC) Layer.

   ```bash
   # tc -s filter show dev p5p1_1 ingress
   ...
   filter block 94 protocol ip pref 3 flower chain 5
   filter block 94 protocol ip pref 3 flower chain 5 handle 0x2
   eth_type ipv4
   src_ip 172.0.0.1
   ip_flags nofrag
   in_hw in_hw_count 1
   action order 1: mirred (Egress Redirect to device eth4) stolen
   index 3 ref 1 bind 1 installed 364 sec used 0 sec
   Action statistics:
   Sent 253991716224 bytes 169534118 pkt (dropped 0, overlimits 0 requeues 0)
   Sent software 43711874200 bytes 30161170 pkt
   Sent hardware 210279842024 bytes 139372948 pkt
   backlog 0b 0p requeues 0
   cookie 8beddad9a0430f0457e7e78db6e0af48
   no_percpu
   ```

3. Examine the `in_hw` flags and the statistics in this output. The word `hardware` indicates that the hardware processes the network traffic. If you use `tc-policy=none`, you can check this output or a tcpdump to investigate when hardware or software handles the packets. You can see a corresponding log message in `dmesg` or in `ovs-vswitch.log` when the driver is unable to offload packets.

4. For Mellanox, as an example, the log entries resemble syndrome messages in `dmesg`.

   ```text
   [13232.860484] mlx5_core 0000:3b:00.0: mlx5_cmd_check:756:(pid 131368):
   SET_FLOW_TABLE_ENTRY(0x936) op_mod(0x0) failed, status bad parameter(0x3),
   syndrome (0x6b1266)
   ```

   In this example, the error code (0x6b1266) represents the following behavior:

   ```text
   0x6B1266 | set_flow_table_entry: pop vlan and forward to uplink is not allowed
   ```

Validating systems

Validate your system with the following procedure.

Procedure

1. Ensure SR-IOV and VT-d are enabled on the system.
2. Enable IOMMU in Linux by adding `intel_iommu=on` to kernel parameters, for example, using GRUB.

Limitations

You cannot use the OVS firewall driver with HW offload because the connection tracking properties of the flows are unsupported in the offload path in OVS 2.11.

6.9. DEBUGGING HW OFFLOAD FLOW

You can use the following procedure if you encounter the following message in the `ovs-vswitch.log` file:

```
2020-01-31T06:22:11.257Z|00473|dpif_netlink(handler402)|ERR|failed to offload flow: Operation not supported: p6p1_5
```

Procedure

1. To enable logging on the offload modules and to get additional log information for this failure, use the following commands on the Compute node:

   ```
   ovs-appctl vlog/set dpif_netlink:file:dbg
   # Module name changed recently (check based on the version used
   ovs-appctl vlog/set netdev_tc_offloads:file:dbg [OR] ovs-appctl vlog/set
   netdev_offload_tc:file:dbg
   ovs-appctl vlog/set tc:file:dbg
   ```

2. Inspect the `ovs-vswitchd` logs again to see additional details about the issue. In the following example logs, the offload failed because of an unsupported attribute mark.

   ```
   2020-01-31T06:22:11.218Z|00471|dpif_netlink(handler402)|DBG|system@ovs-system:
   put[create] ufid:61bd016e-eb89-44fc-a17e-958bc8e45fda
   recirc_id(0),dp_hash(0/0),skb_priority(0/0),in_port(7),skb_mark(0),ct_state(0/0),ct_zone(0/0),ct_mark(0/0),ct_label(0/0),eth(src=fa:16:3e:d2:f5:f3,dst=fa:16:3e:c4:a3:eb),eth_type(0x0800),ipv4(src=10.1.1.8/0.0.0.0,dst=10.1.1.31/0.0.0.0,proto=1/0,tos=0/0x3,ttl=64/0,frag=no),icmp(type=0/0,code=0/0),actions:set(tunnel(tun_id=0x3d,src=10.10.141.107,dst=10.10.141.124,ttl=64,tp_dst=4789,flags(df|key))),6
   ```

   ```
   2020-01-31T06:22:11.253Z|00472|netdev_tc_offloads(handler402)|DBG|offloading attribute pkt_mark isn't supported
   ```

   ```
   2020-01-31T06:22:11.257Z|00473|dpif_netlink(handler402)|ERR|failed to offload flow: Operation not supported: p6p1_5
   ```

Debugging Mellanox NICs

Mellanox has provided a system information script, similar to a Red Hat SOS report.


When you run this command, you create a zip file of the relevant log information, which is useful for support cases.

Procedure
You can run this system information script with the following command:

```
# ./sysinfo-snapshot.py --asap --asap_tc --ibdiagnet --openstack
```

You can also install Mellanox Firmware Tools (MFT), mlxconfig, mlxlink and the OpenFabrics Enterprise Distribution (OFED) drivers.

**Useful CLI commands**

Use the `ethtool` utility with the following options to gather diagnostic information:

- `ethtool -l <uplink representor>`: View the number of channels
- `ethtool -I <uplink/VFs>`: Check statistics
- `ethtool -i <uplink rep>`: View driver information
- `ethtool -g <uplink rep>`: Check ring sizes
- `ethtool -k <uplink/VFs>`: View enabled features

Use the `tcpdump` utility at the representor and PF ports to similarly check traffic flow.

- Any changes you make to the link state of the representor port, affect the VF link state also.
- Representor port statistics present VF statistics also.

Use the below commands to get useful diagnostic information:

```
$ ovs-appctl dpctl/dump-flows -m type=offloaded
$ ovs-appctl dpctl/dump-flows -m
$ tc filter show dev ens1_0 ingress
$ tc -s filter show dev ens1_0 ingress
$ tc monitor
```

6.10. **DEPLOYING AN INSTANCE FOR SR-IOV**

Use host aggregates to separate high performance compute hosts. For information on creating host aggregates and associated flavors for scheduling see [Creating host aggregates](#).

**NOTE**

Pinned CPU instances can be located on the same Compute node as unpinned instances. For more information, see [Configuring CPU pinning on Compute nodes](#) in the [Configuring the Compute Service for Instance Creation](#) guide.

Deploy an instance for single root I/O virtualization (SR-IOV) by performing the following steps:

1. Create a flavor.

```
$ openstack flavor create <flavor> --ram <MB> --disk <GB> --vcpus <#>
```
You can specify the NUMA affinity policy for PCI passthrough devices and SR-IOV interfaces by adding the extra spec `hw:pci_numa_affinity_policy` to your flavor. For more information, see Flavor metadata in the Configuring the Compute Service for Instance Creation guide.

2. Create the network.

   $ openstack network create net1 --provider-physical-network tenant --provider-network-type vlan --provider-segment <VLAN-ID>

   $ openstack subnet create subnet1 --network net1 --subnet-range 192.0.2.0/24 --dhcp

3. Create the port.

   - Use vnic-type `direct` to create an SR-IOV virtual function (VF) port.
     
     $ openstack port create --network net1 --vnic-type direct sriov_port

   - Use the following command to create a virtual function with hardware offload. You must be an admin user to set `--binding-profile`.
     
     $ openstack port create --network net1 --vnic-type direct --binding-profile '{"capabilities": ["switchdev"]}' sriov_hwoffload_port

   - Use vnic-type `direct-physical` to create an SR-IOV physical function (PF) port that is dedicated to a single instance. This PF port is a Networking service (neutron) port but is not controlled by the Networking service, and is not visible as a network adapter because it is a PCI device that is passed through to the instance.
     
     $ openstack port create --network net1 --vnic-type direct-physical sriov_port

4. Deploy an instance.

   $ openstack server create --flavor <flavor> --image <image> --nic port-id=<id> <instance name>

### 6.11. CREATING HOST AGGREGATES

For better performance, deploy guests that have CPU pinning and huge pages. You can schedule high performance instances on a subset of hosts by matching aggregate metadata with flavor metadata.

1. You can configure the `AggregateInstanceExtraSpecsFilter` value, and other necessary filters, through the heat parameter `NovaSchedulerEnabledFilters` under `parameter_defaults` in your deployment templates.

```
parameter_defaults:
  NovaSchedulerEnabledFilters:
    - AggregateInstanceExtraSpecsFilter
    - AvailabilityZoneFilter
    - ComputeFilter
    - ComputeCapabilitiesFilter
    - ImagePropertiesFilter
```
- ServerGroupAntiAffinityFilter
- ServerGroupAffinityFilter
- PciPassthroughFilter
- NUMATopologyFilter

**NOTE**

To add this parameter to the configuration of an existing cluster, you can add it to the heat templates, and run the original deployment script again.

2. Create an aggregate group for SR-IOV, and add relevant hosts. Define metadata, for example, `sriov=true`, that matches defined flavor metadata.

```bash
# openstack aggregate create sriov_group
# openstack aggregate add host sriov_group compute-sriov-0.localdomain
# openstack aggregate set --property sriov=true sriov_group
```

3. Create a flavor.

```bash
# openstack flavor create <flavor> --ram <MB> --disk <GB> --vcpus <#>
```

4. Set additional flavor properties. Note that the defined metadata, `sriov=true`, matches the defined metadata on the SR-IOV aggregate.

```bash
# openstack flavor set --property sriov=true --property hw:cpu_policy=dedicated --property hw:mem_page_size=1GB <flavor>
```
CHAPTER 7. PLANNING YOUR OVS-DPDK DEPLOYMENT

To optimize your Open vSwitch with Data Plane Development Kit (OVS-DPDK) deployment for NFV, you should understand how OVS-DPDK uses the Compute node hardware (CPU, NUMA nodes, memory, NICs) and the considerations for determining the individual OVS-DPDK parameters based on your Compute node.

**IMPORTANT**

When using OVS-DPDK and the OVS native firewall (a stateful firewall based on conntrack), you can track only packets that use ICMPv4, ICMPv6, TCP, and UDP protocols. OVS marks all other types of network traffic as invalid.

**IMPORTANT**

Red Hat does not support the use of OVS-DPDK for non-NFV workloads. If you need OVS-DPDK functionality for non-NFV workloads, contact your Technical Account Manager (TAM) or open a customer service request case to discuss a Support Exception and other options. To open a customer service request case, go to Create a case and choose Account > Customer Service Request.

For a high-level introduction to CPUs and NUMA topology, see NFV performance considerations.

7.1. OVS-DPDK WITH CPU PARTITIONING AND NUMA TOPOLOGY

OVS-DPDK partitions the hardware resources for host, guests, and itself. The OVS-DPDK Poll Mode Drivers (PMDs) run DPDK active loops, which require dedicated CPU cores. Therefore you must allocate some CPUs, and huge pages, to OVS-DPDK.

A sample partitioning includes 16 cores per NUMA node on dual-socket Compute nodes. The traffic requires additional NICs because you cannot share NICs between the host and OVS-DPDK.

**NOTE**

You must reserve DPDK PMD threads on both NUMA nodes, even if a NUMA node does not have an associated DPDK NIC.
For optimum OVS-DPDK performance, reserve a block of memory local to the NUMA node. Choose NICs associated with the same NUMA node that you use for memory and CPU pinning. Ensure that both bonded interfaces are from NICs on the same NUMA node.

### 7.2. CALCULATING OVS-DPDK PARAMETERS

This section describes how OVS-DPDK uses parameters within the director `network_environment.yaml` heat templates to configure the CPU and memory for optimum performance. Use this information to evaluate the hardware support on your Compute nodes and how to partition the hardware to optimize your OVS-DPDK deployment.

**NOTE**

Always pair CPU sibling threads, or logical CPUs, together in the physical core when allocating CPU cores.

For details on how to determine the CPU and NUMA nodes on your Compute nodes, see Discovering your NUMA node topology. Use this information to map CPU and other parameters to support the host, guest instance, and OVS-DPDK process needs.

#### 7.2.1. CPU parameters

OVS-DPDK uses the following parameters for CPU partitioning:

**OvsPmdCoreList**

Provides the CPU cores that are used for the DPDK poll mode drivers (PMD). Choose CPU cores that are associated with the local NUMA nodes of the DPDK interfaces. Use `OvsPmdCoreList` for the `pmd-cpu-mask` value in OVS. Use the following recommendations for `OvsPmdCoreList`:

- Pair the sibling threads together.
- Performance depends on the number of physical cores allocated for this PMD Core list. On the NUMA node which is associated with DPDK NIC, allocate the required cores.
- For NUMA nodes with a DPDK NIC, determine the number of physical cores required based on the performance requirement, and include all the sibling threads or logical CPUs for each physical core.
- For NUMA nodes without DPDK NICs, allocate the sibling threads or logical CPUs of any physical core except the first physical core of the NUMA node.

**NOTE**

You must reserve DPDK PMD threads on both NUMA nodes, even if a NUMA node does not have an associated DPDK NIC.

**NovaComputeCpuDedicatedSet**

A comma-separated list or range of physical host CPU numbers to which processes for pinned instance CPUs can be scheduled. For example, `NovaComputeCpuDedicatedSet: [4-12,^8,15]` reserves cores from 4-12 and 15, excluding 8.

- Exclude all cores from the `OvsPmdCoreList`.

Red Hat OpenStack Platform 17.1-Beta Configuring network functions virtualization
• Include all remaining cores.
• Pair the sibling threads together.

**NovaComputeCpuSharedSet**
A comma-separated list or range of physical host CPU numbers used to determine the host CPUs for instance emulator threads.

**IsolCpusList**
A set of CPU cores isolated from the host processes. **IsolCpusList** is the isolated_cores value in the cpu-partitioning-variable.conf file for the tuned-profiles-cpu-partitioning component. Use the following recommendations for **IsolCpusList**:

• Match the list of cores in **OvsPmdCoreList** and **NovaComputeCpuDedicatedSet**.
• Pair the sibling threads together.

**DerivePciWhitelistEnabled**
To reserve virtual functions (VF) for VMs, use the **NovaPCIPassthrough** parameter to create a list of VFs passed through to Nova. VFs excluded from the list remain available for the host.
For each VF in the list, populate the address parameter with a regular expression that resolves to the address value.

The following is an example of the manual list creation process. If NIC partitioning is enabled in a device named **eno2**, list the PCI addresses of the VFs with the following command:

```
[tripleo-admin@compute-0 ~]$ ls -lh /sys/class/net/eno2/device/ | grep virtfn
lnwxrwxrwx. 1 root root 0 Apr 16 09:58 virtfn0 -> ../0000:18:06.0
lnwxrwxrwx. 1 root root 0 Apr 16 09:58 virtfn1 -> ../0000:18:06.1
lnwxrwxrwx. 1 root root 0 Apr 16 09:58 virtfn2 -> ../0000:18:06.2
lnwxrwxrwx. 1 root root 0 Apr 16 09:58 virtfn3 -> ../0000:18:06.3
lnwxrwxrwx. 1 root root 0 Apr 16 09:58 virtfn4 -> ../0000:18:06.4
lnwxrwxrwx. 1 root root 0 Apr 16 09:58 virtfn5 -> ../0000:18:06.5
lnwxrwxrwx. 1 root root 0 Apr 16 09:58 virtfn6 -> ../0000:18:06.6
lnwxrwxrwx. 1 root root 0 Apr 16 09:58 virtfn7 -> ../0000:18:06.7
```

In this case, the VFs 0, 4, and 6 are used by **eno2** for NIC Partitioning. Manually configure **NovaPCIPassthrough** to include VFs 1-3, 5, and 7, and consequently exclude VFs 0, 4, and 6, as in the following example:

```
NovaPCIPassthrough:
- physical_network: "sriovnet2"
  address: {"domain": ".*", "bus": "18", "slot": "06", "function": "[1-3]"}
- physical_network: "sriovnet2"
  address: {"domain": ".*", "bus": "18", "slot": "06", "function": "[5]"}
- physical_network: "sriovnet2"
  address: {"domain": ".*", "bus": "18", "slot": "06", "function": "[7]"}
```

### 7.2.2. Memory parameters

OVS-DPDK uses the following memory parameters:

**OvsDpdkMemoryChannels**
Maps memory channels in the CPU per NUMA node. **OvsDpdkMemoryChannels** is the **other_config:dpdk-extra="-n <value>"** value in OVS. Observe the following recommendations for **OvsDpdkMemoryChannels**:

- Use `dmidecode -t memory` or your hardware manual to determine the number of memory channels available.
- Use `ls /sys/devices/system/node/node*/-d` to determine the number of NUMA nodes.
- Divide the number of memory channels available by the number of NUMA nodes.

**NovaReservedHostMemory**

Reserves memory in MB for tasks on the host. **NovaReservedHostMemory** is the **reserved_host_memory_mb** value for the Compute node in **nova.conf**. Observe the following recommendation for **NovaReservedHostMemory**:

- Use the static recommended value of 4096 MB.

**OvsDpdkSocketMemory**

Specifies the amount of memory in MB to pre-allocate from the hugepage pool, per NUMA node. **OvsDpdkSocketMemory** is the **other_config:dpdk-socket-mem** value in OVS. Observe the following recommendations for **OvsDpdkSocketMemory**:

- Provide as a comma-separated list.
- For a NUMA node without a DPDK NIC, use the static recommendation of 1024 MB (1GB)
- Calculate the **OvsDpdkSocketMemory** value from the MTU value of each NIC on the NUMA node.
- The following equation approximates the value for **OvsDpdkSocketMemory**:
  
  \[
  \text{MEMORY\_REQT\_PER\_MTU} = (\text{ROUNDUP\_PER\_MTU} + 800) \times (4096 \times 64) \text{ Bytes}
  \]
  
  - 800 is the overhead value.
  - 4096 * 64 is the number of packets in the mempool.
- Add the **MEMORY\_REQT\_PER\_MTU** for each of the MTU values set on the NUMA node and add another 512 MB as buffer. Round the value up to a multiple of 1024.

**Sample Calculation - MTU 2000 and MTU 9000**

DPDK NICs dpdk0 and dpdk1 are on the same NUMA node 0, and configured with MTUs 9000, and 2000 respectively. The sample calculation to derive the memory required is as follows:

1. Round off the MTU values to the nearest multiple of 1024 bytes.
   
   The MTU value of 9000 becomes 9216 bytes.
   The MTU value of 2000 becomes 2048 bytes.

2. Calculate the required memory for each MTU value based on these rounded byte values.
   
   Memory required for 9000 MTU = (9216 + 800) * (4096*64) = 2625634304
   Memory required for 2000 MTU = (2048 + 800) * (4096*64) = 746586112
3. Calculate the combined total memory required, in bytes.

\[
2625634304 + 746586112 + 536870912 = 3909091328 \text{ bytes.}
\]

This calculation represents (Memory required for MTU of 9000) + (Memory required for MTU of 2000) + (512 MB buffer).

4. Convert the total memory required into MB.

\[
3909091328 / (1024*1024) = 3728 \text{ MB.}
\]

5. Round this value up to the nearest 1024.

3724 MB rounds up to 4096 MB.

6. Use this value to set `OvsDpdkSocketMemory`.

`OvsDpdkSocketMemory`: "4096,1024"

**Sample Calculation - MTU 2000**

DPDK NICs dpdk0 and dpdk1 are on the same NUMA node 0, and each are configured with MTUs of 2000. The sample calculation to derive the memory required is as follows:

1. Round off the MTU values to the nearest multiple of 1024 bytes.

   The MTU value of 2000 becomes 2048 bytes.

2. Calculate the required memory for each MTU value based on these rounded byte values.

   Memory required for 2000 MTU = (2048 + 800) * (4096*64) = 746586112

3. Calculate the combined total memory required, in bytes.

\[
746586112 + 536870912 = 1283457024 \text{ bytes.}
\]

This calculation represents (Memory required for MTU of 2000) + (512 MB buffer).

4. Convert the total memory required into MB.

\[
1283457024 / (1024*1024) = 1224 \text{ MB.}
\]

5. Round this value up to the nearest multiple of 1024.

1224 MB rounds up to 2048 MB.

6. Use this value to set `OvsDpdkSocketMemory`.

`OvsDpdkSocketMemory`: "2048,1024"

### 7.2.3. Networking parameters
OvsDpdkDriverType
Sets the driver type used by DPDK. Use the default value of \texttt{vfio-pci}.

NeutronDatapathType
Datapath type for OVS bridges. DPDK uses the default value of \texttt{netdev}.

NeutronVhostuserSocketDir
Sets the vhost-user socket directory for OVS. Use \texttt{/var/lib/vhost_sockets} for vhost client mode.

7.2.4. Other parameters

NovaSchedulerEnabledFilters
Provides an ordered list of filters that the Compute node uses to find a matching Compute node for a requested guest instance.

VhostuserSocketGroup
Sets the vhost-user socket directory group. The default value is \texttt{qemu}. Set \texttt{VhostuserSocketGroup} to \texttt{hugetlbfs} so that the \texttt{ovs-vswitchd} and \texttt{qemu} processes can access the shared huge pages and unix socket that configures the virtio-net device. This value is role-specific and should be applied to any role leveraging OVS-DPDK.

KernelArgs
Provides multiple kernel arguments to \texttt{/etc/default/grub} for the Compute node at boot time. Add the following values based on your configuration:

- \texttt{hugepagesz}: Sets the size of the huge pages on a CPU. This value can vary depending on the CPU hardware. Set to 1G for OVS-DPDK deployments (\texttt{default_hugepagesz=1GB} \texttt{hugepagesz=1G}). Use this command to check for the \texttt{pdpe1gb} CPU flag that confirms your CPU supports 1G.

\begin{verbatim}
 lshw -class processor | grep pdpe1gb
\end{verbatim}

- \texttt{hugepages count}: Sets the number of huge pages available based on available host memory. Use most of your available memory, except \texttt{NovaReservedHostMemory}. You must also configure the huge pages count value within the flavor of your Compute nodes.

- \texttt{iommu}: For Intel CPUs, add \texttt{"intel_iommu=on iommu=pt"}

- \texttt{isolcpus}: Sets the CPU cores for tuning. This value matches \texttt{IsolCpusList}.

For more information about CPU isolation, see the Red Hat Knowledgebase solution OpenStack CPU isolation guidance for RHEL 8 and RHEL 9

7.2.5. Instance extra specifications

Before deploying instances in an NFV environment, create a flavor that utilizes CPU pinning, huge pages, and emulator thread pinning.

\texttt{hw:cpu_policy}
When this parameter is set to \texttt{dedicated}, the guest uses pinned CPUs. Instances created from a flavor with this parameter set have an effective overcommit ratio of 1:1. The default value is \texttt{shared}.

\texttt{hw:mem_page_size}
Set this parameter to a valid string of a specific value with standard suffix (For example, \texttt{4KB}, \texttt{8MB}, or \texttt{1GB}). Use 1GB to match the \texttt{hugepagesz} boot parameter. Calculate the number of huge pages
available for the virtual machines by subtracting OvsDpdkSocketMemory from the boot parameter. The following values are also valid:

- **small** (default) - The smallest page size is used
- **large** - Only use large page sizes. (2MB or 1GB on x86 architectures)
- **any** - The compute driver can attempt to use large pages, but defaults to small if none available.

**hw:emulator_threads_policy**

Set the value of this parameter to `share` so that emulator threads are locked to CPUs that you’ve identified in the heat parameter, NovaComputeCpuSharedSet. If an emulator thread is running on a vCPU with the poll mode driver (PMD) or real-time processing, you can experience negative effects, such as packet loss.

### 7.3. TWO NUMA NODE EXAMPLE OVS-DPDK DEPLOYMENT

The Compute node in the following example includes two NUMA nodes:

- **NUMA 0** has cores 0-7. The sibling thread pairs are (0,1), (2,3), (4,5), and (6,7)
- **NUMA 1** has cores 8-15. The sibling thread pairs are (8,9), (10,11), (12,13), and (14,15).
- Each NUMA node connects to a physical NIC, namely NIC1 on NUMA 0, and NIC2 on NUMA 1.

![NUMA NODE 0](image1.png) ![NUMA NODE 1](image2.png)

**NOTE**

Reserve the first physical cores or both thread pairs on each NUMA node (0,1 and 8,9) for non-datapath DPDK processes.

This example also assumes a 1500 MTU configuration, so the OvsDpdkSocketMemory is the same for all use cases:

**OvsDpdkSocketMemory: "1024,1024"**

**NIC 1 for DPDK, with one physical core for PMD**

In this use case, you allocate one physical core on NUMA 0 for PMD. You must also allocate one physical core on NUMA 1, even though DPDK is not enabled on the NIC for that NUMA node. The remaining cores are allocated for guest instances. The resulting parameter settings are:
NIC 1 for DPDK, with two physical cores for PMD

In this use case, you allocate two physical cores on NUMA 0 for PMD. You must also allocate one physical core on NUMA 1, even though DPDK is not enabled on the NIC for that NUMA node. The remaining cores are allocated for guest instances. The resulting parameter settings are:

OvsPmdCoreList: "2,3,10,11"
NovaComputeCpuDedicatedSet: "4,5,6,7,12,13,14,15"

NIC 2 for DPDK, with one physical core for PMD

In this use case, you allocate one physical core on NUMA 1 for PMD. You must also allocate one physical core on NUMA 0, even though DPDK is not enabled on the NIC for that NUMA node. The remaining cores are allocated for guest instances. The resulting parameter settings are:

OvsPmdCoreList: "2,3,10,11"
NovaComputeCpuDedicatedSet: "6,7,12,13,14,15"

NIC 2 for DPDK, with two physical cores for PMD

In this use case, you allocate two physical cores on NUMA 1 for PMD. You must also allocate one physical core on NUMA 0, even though DPDK is not enabled on the NIC for that NUMA node. The remaining cores are allocated for guest instances. The resulting parameter settings are:

OvsPmdCoreList: "2,3,10,11,12,13"
NovaComputeCpuDedicatedSet: "4,5,6,7,14,15"

NIC 1 and NIC2 for DPDK, with two physical cores for PMD

In this use case, you allocate two physical cores on each NUMA node for PMD. The remaining cores are allocated for guest instances. The resulting parameter settings are:

OvsPmdCoreList: "2,3,4,5,10,11,12,13"
NovaComputeCpuDedicatedSet: "6,7,14,15"

7.4. TOPOLOGY OF AN NFV OVS-DPDK DEPLOYMENT

This example deployment shows an OVS-DPDK configuration and consists of two virtual network functions (VNFs) with two interfaces each:

- The management interface, represented by mgt.
- The data plane interface.

In the OVS-DPDK deployment, the VNFs operate with inbuilt DPDK that supports the physical interface. OVS-DPDK enables bonding at the vSwitch level. For improved performance in your OVS-DPDK deployment, it is recommended that you separate kernel and OVS-DPDK NICs. To separate the management (mgt) network, connected to the Base provider network for the virtual machine, ensure you have additional NICs. The Compute node consists of two regular NICs for the Red Hat OpenStack Platform API management that can be reused by the Ceph API but cannot be shared with any OpenStack project.
NFV OVS-DPDK topology

The following image shows the topology for OVS-DPDK for NFV. It consists of Compute and Controller nodes with 1 or 10 Gbps NICs, and the director node.
Configuring network functions virtualization
CHAPTER 8. CONFIGURING AN OVS-DPDK DEPLOYMENT

This section deploys OVS-DPDK within the Red Hat OpenStack Platform environment. The overcloud usually consists of nodes in predefined roles such as Controller nodes, Compute nodes, and different storage node types. Each of these default roles contains a set of services defined in the core heat templates on the director node.

You must install and configure the undercloud before you can deploy the overcloud. See the Installing and managing Red Hat OpenStack Platform with director Guide for details.

NOTE

Do not manually edit or change `isolated_cores` or other values in `etc/tuned/cpu-partitioning-variables.conf` that the director heat templates modify.

8.1. OVS-DPDK TOPOLOGY

With Red Hat OpenStack Platform, you can create custom deployment roles, using the composable roles feature to add or remove services from each role. For more information on Composable Roles, see Composable services and custom roles in Advanced Overcloud Customization.

This image shows an example OVS-DPDK topology with two bonded ports for the control plane and data plane:
To configure OVS-DPDK, perform the following tasks:

- If you use composable roles, copy and modify the `roles_data.yaml` file to add the custom role for OVS-DPDK.
- Update the appropriate `network-environment.yaml` file to include parameters for kernel arguments, and DPDK arguments.
- Update the `compute.yaml` file to include the bridge for DPDK interface parameters.
- Update the `controller.yaml` file to include the same bridge details for DPDK interface parameters.
- Run the `overcloud_deploy.sh` script to deploy the overcloud with the DPDK parameters.
NOTE
This guide provides examples for CPU assignments, memory allocation, and NIC configurations that can vary from your topology and use case. For more information on hardware and configuration options, see: Introduction to network functions virtualization and Chapter 2, Hardware requirements.

Prerequisites

- OVS 2.10
- DPDK 17
- A supported NIC. To view the list of supported NICs for NFV, see Section 2.1, “Tested NICs”.

NOTE
The Red Hat OpenStack Platform operates in OVS client mode for OVS-DPDK deployments.

8.2. SETTING THE MTU VALUE FOR OVS-DPDK INTERFACES

Red Hat OpenStack Platform supports jumbo frames for OVS-DPDK. To set the maximum transmission unit (MTU) value for jumbo frames you must:

- Set the global MTU value for networking in the network-environment.yaml file.
- Set the physical DPDK port MTU value in the compute.yaml file. This value is also used by the vhost user interface.
- Set the MTU value within any guest instances on the Compute node to ensure that you have a comparable MTU value from end to end in your configuration.

NOTE
VXLAN packets include an extra 50 bytes in the header. Calculate your MTU requirements based on these additional header bytes. For example, an MTU value of 9000 means the VXLAN tunnel MTU value is 8950 to account for these extra bytes.

NOTE
You do not need any special configuration for the physical NIC because the NIC is controlled by the DPDK PMD, and has the same MTU value set by the compute.yaml file. You cannot set an MTU value larger than the maximum value supported by the physical NIC.

To set the MTU value for OVS-DPDK interfaces:


```
parameter_defaults:
  # MTU global configuration
  NeutronGlobalPhysnetMtu: 9000
```
NOTE

Ensure that the OvsDpdkSocketMemory value in the network-environment.yaml file is large enough to support jumbo frames. For details, see Section 7.2.2, “Memory parameters”.

2. Set the MTU value on the bridge to the Compute node in the controller.yaml file:

```yaml
- type: ovs_bridge
  name: br-link0
  use_dhcp: false
  members:
    - type: interface
      name: nic3
      mtu: 9000
```

3. Set the MTU values for an OVS-DPDK bond in the compute.yaml file:

```yaml
- type: ovs_user_bridge
  name: br-link0
  use_dhcp: false
  members:
    - type: ovs_dpdk_bond
      name: dpdkbond0
      mtu: 9000
      rx_queue: 2
      members:
        - type: ovs_dpdk_port
          name: dpdk0
          mtu: 9000
          members:
            - type: interface
              name: nic4
            - type: ovs_dpdk_port
              name: dpdk1
              mtu: 9000
              members:
                - type: interface
                  name: nic5
```

8.3. CONFIGURING A FIREWALL FOR SECURITY GROUPS

Dataplane interfaces require high performance in a stateful firewall. To protect these interfaces, consider deploying a telco-grade firewall as a virtual network function (VNF).

To configure control plane interfaces in an ML2/OVS deployment, set the NeutronOVSFirewallDriver parameter to openvswitch. To use the flow-based OVS firewall driver, modify the network-environment.yaml file under parameter_defaults. In an OVN deployment, you can implement security groups with Access Control Lists (ACL).

You cannot use the OVS firewall driver with HW offload because the connection tracking properties of the flows are unsupported in the offload path.
Example:

parameter_defaults:
NeutronOVSFirewallDriver: openvswitch

Use the `openstack port set` command to disable the OVS firewall driver for dataplane interfaces.

Example:

```
openstack port set --no-security-group --disable-port-security ${PORT}
```

### 8.4. SETTING MULTIQUEUE FOR OVS-DPDK INTERFACES

**NOTE**

Multiqueue is experimental, and only supported with manual queue pinning.

**Procedure**

- To set the same number of queues for interfaces in OVS-DPDK on the Compute node, modify the `compute.yaml` file:

```
- type: ovs_user_bridge
  name: br-link0
  use_dhcp: false
  members:
  - type: ovs_dpdk_bond
    name: dpdkbond0
    mtu: 9000
    rx_queue: 2
    members:
      - type: ovs_dpdk_port
        name: dpdk0
        mtu: 9000
        members:
          - type: interface
            name: nic4
      - type: ovs_dpdk_port
        name: dpdk1
        mtu: 9000
        members:
          - type: interface
            name: nic5
```

### 8.5. CONFIGURING OVS PMD AUTO LOAD BALANCE

**IMPORTANT**

This feature is available in this release as a *Technology Preview*, and therefore is not fully supported by Red Hat. It should only be used for testing, and should not be deployed in a production environment. For more information about Technology Preview features, see *Scope of Coverage Details*. 
You can use Open vSwitch (OVS) Poll Mode Driver (PMD) threads to perform the following tasks for user space context switching:

- Continuous polling of input ports for packets.
- Classifying received packets.
- Executing actions on the packets after classification.

You can configure your RHOSP deployment to automatically load balance the OVS PMD threads with the following parameters:

- **OvsPmdAutoLb**
- **OvsPmdLoadThreshold**
- **OvsPmdImprovementThreshold**
- **OvsPmdRebalInterval**

**Procedure**

1. Change the value of the **OvsPmdAutoLb** parameter to `true` to enable automatic PMD load balancing:

   ```
   parameter_defaults:
   OvsPmdAutoLb: true
   ```

2. Specify the percentage limit of used cycles that triggers the PMD load balance with the **OvsPmdLoadThreshold** parameter:

   ```
   parameter_defaults:
   OvsPmdAutoLb: true
   OvsPmdLoadThreshold: <load_threshold>
   ```

   Replace `<load_threshold>` with a number between 0 and 100, to represent the minimum percentage of PMD thread load that triggers the automatic load balancing.

3. Specify the minimum percentage of evaluated improvement across the non-isolated PMD threads that triggers a PMD Auto Load Balance **OvsPmdImprovementThreshold** parameter:

   ```
   parameter_defaults:
   OvsPmdAutoLb: true
   OvsPmdLoadThreshold: <load_threshold>
   OvsPmdImprovementThreshold: <improvement_threshold>
   ```

   Replace `<improvement_threshold>` with a number between 0 and 100, to represent the minimum percentage of evaluated improvement that triggers the automatic load balancing.

4. Specify the minimum time between two consecutive PMD Auto Load Balance operations with the **OvsPmdRebalInterval** parameter:

   ```
   parameter_defaults:
   OvsPmdAutoLb: true
   OvsPmdLoadThreshold: <load_threshold>
   ```
5. Add your OVS PMD environment file to the stack with your other environment files, and deploy the overcloud:

(undercloud)$ openstack overcloud deploy --templates \
-e [your environment files] \
-e /home/stack/templates/auto_ovs_pmd.yaml

8.6. KNOWN LIMITATIONS

Observe the following limitations when configuring OVS-DPDK with Red Hat OpenStack Platform for NFV:

- Use Linux bonds for non-DPDK traffic, and control plane networks, such as Internal, Management, Storage, Storage Management, and Tenant. Ensure that both the PCI devices used in the bond are on the same NUMA node for optimum performance. Neutron Linux bridge configuration is not supported by Red Hat.

- You require huge pages for every instance running on the hosts with OVS-DPDK. If huge pages are not present in the guest, the interface appears but does not function.

- With OVS-DPDK, there is a performance degradation of services that use tap devices, such as Distributed Virtual Routing (DVR). The resulting performance is not suitable for a production environment.

- When using OVS-DPDK, all bridges on the same Compute node must be of type `ovs_user_bridge`. The director may accept the configuration, but Red Hat OpenStack Platform does not support mixing `ovs_bridge` and `ovs_user_bridge` on the same node.

8.7. CREATING A FLAVOR AND DEPLOYING AN INSTANCE FOR OVS-DPDK

After you configure OVS-DPDK for your Red Hat OpenStack Platform deployment with NFV, you can create a flavor, and deploy an instance using the following steps:

1. Create an aggregate group, and add relevant hosts for OVS-DPDK. Define metadata, for example `dpdk=true`, that matches defined flavor metadata.

   ```bash
   # openstack aggregate create dpdk_group
   # openstack aggregate add host dpdk_group [compute-host]
   # openstack aggregate set --property dpdk=true dpdk_group
   
   NOTE
   Pinned CPU instances can be located on the same Compute node as unpinned instances. For more information, see Configuring CPU pinning on Compute nodes in the Configuring the Compute Service for Instance Creation guide.
   
   2. Create a flavor.
# openstack flavor create <flavor> --ram <MB> --disk <GB> --vcpus <#>

3. Set flavor properties. Note that the defined metadata, `dpdk=true`, matches the defined metadata in the DPDK aggregate.

```
# openstack flavor set <flavor> --property dpdk=true --property hw:cpu_policy=dedicated --property hw:mem_page_size=1GB --property hw:emulator_threads_policy= isol ate
```

For details about the emulator threads policy for performance improvements, see Configuring emulator threads in the Configuring the Compute Service for Instance Creation guide.

4. Create the network.

```
# openstack network create net1 --provider-physical-network tenant --provider-network-type vlan --provider-segment <VLAN-ID>
# openstack subnet create subnet1 --network net1 --subnet-range 192.0.2.0/24 --dhcp
```

5. Optional: If you use multiqueue with OVS-DPDK, set the `hw_vif_multiqueue_enabled` property on the image that you want to use to create a instance:

```
# openstack image set --property hw_vif_multiqueue_enabled=true <image>
```

6. Deploy an instance.

```
# openstack server create --flavor <flavor> --image <glance image> --nic net-id=<network ID> <server_name>
```

### 8.8. TROUBLESHOOTING THE OVS-DPDK CONFIGURATION

This section describes the steps to troubleshoot the OVS-DPDK configuration.

1. Review the bridge configuration, and confirm that the bridge has `datapath_type=netdev`.

```
# ovs-vsctl list bridge br0
_uuid               : bdce0825-e263-4d15-b256-f01222df96f3
auto_attach         : []
controller          : []
datapath_id         : "00002608cebd154d"
datapath_type       : netdev
datapath_version    : "<built-in>"
external_ids        : {}
fail_mode           : []
flood_vlans         : []
flow_tables         : {}
ipfix               : []
mcast_snooping_enable: false
mirrors             : []
name                : "br0"
netflow             : []
other_config        : {}
ports               : [52725b91-de7f-41e7-bb49-3b7e50354138]
protocols           : []
rstp_enable         : false
```

rstp_status : {}
sflow : []
status : {}
stp_enable : false

2. Optionally, you can view logs for errors, such as if the container fails to start.

   # less /var/log/containers/neutron/openvswitch-agent.log

3. Confirm that the Poll Mode Driver CPU mask of the **ovs-dpdk** is pinned to the CPUs. In case of hyper threading, use sibling CPUs.
   For example, to check the sibling of **CPU4**, run the following command:

   # cat /sys/devices/system/cpu/cpu4/topology/thread_siblings_list
   4,20

   The sibling of **CPU4** is **CPU20**, therefore proceed with the following command:

   # ovs-vsctl set Open_vSwitch . other_config:pmd-cpu-mask=0x100010

Display the status:

   # tuna -t ovs-vswitchd -CP
   thread  ctxt_switches pid SCHED_ rtpri  affinity voluntary nonvoluntary  cmd
   3161 OTHER  0     6 765023   614 ovs-vswitchd
   3219 OTHER  0     6    1   0   handler24
   3220 OTHER  0     6    1   0   handler21
   3221 OTHER  0     6    1   0   handler22
   3222 OTHER  0     6    1   0   handler23
   3223 OTHER  0     6    1   0   handler25
   3224 OTHER  0     6    1   0   handler26
   3225 OTHER  0     6    1   0   handler27
   3226 OTHER  0     6    1   0   handler28
   3227 OTHER  0     6    2   0   handler31
   3228 OTHER  0     6    2   4   handler30
   3229 OTHER  0     6    2   5   handler32
   3230 OTHER  0     6 953538   431 revalidator29
   3231 OTHER  0     6 1424258   976 revalidator33
   3232 OTHER  0     6 1424693   836 revalidator34
   3233 OTHER  0     6 951678   503 revalidator36
   3234 OTHER  0     6 1425128   498 revalidator35
   3235 OTHER  0     4 151123   51   pmd37*
   3236 OTHER  0     4 298967    48   pmd38*
   3164 OTHER  0     6 47575   0  dpdk_watchdog3
   3165 OTHER  0     6 237634   0   vhost_thread1
   3166 OTHER  0     6 3665   0    urcu2
CHAPTER 9. TUNING A RED HAT OPENSTACK PLATFORM ENVIRONMENT

9.1. PINNING EMULATOR THREADS

Emulator threads handle interrupt requests and non–blocking processes for virtual machine hardware emulation. These threads float across the CPUs that the guest uses for processing. If threads used for the poll mode driver (PMD) or real-time processing run on these guest CPUs, you can experience packet loss or missed deadlines.

You can separate emulator threads from VM processing tasks by pinning the threads to their own guest CPUs, increasing performance as a result.

9.1.1. Configuring CPUs to host emulator threads

To improve performance, reserve a subset of host CPUs for hosting emulator threads.

Procedure

1. Deploy an overcloud with `NovaComputeCpuSharedSet` defined for a given role. The value of `NovaComputeCpuSharedSet` applies to the `cpu_shared_set` parameter in the `nova.conf` file for hosts within that role.

```bash
parameter_defaults:
  ComputeOvsDpdkParameters:
    NovaComputeCpuSharedSet: "0-1,16-17"
    NovaComputeCpuDedicatedSet: "2-15,18-31"
```

2. Create a flavor to build instances with emulator threads separated into a shared pool.

```bash
openstack flavor create --ram <size_mb> --disk <size_gb> --vcpus <vcpus> <flavor>
```

3. Add the `hw:emulator_threads_policy` extra specification, and set the value to `share`. Instances created with this flavor will use the instance CPUs defined in the `cpu_share_set` parameter in the `nova.conf` file.

```bash
openstack flavor set <flavor> --property hw:emulator_threads_policy=share
```

**NOTE**

You must set the `cpu_share_set` parameter in the `nova.conf` file to enable the share policy for this extra specification. You should use heat for this preferably, as editing `nova.conf` manually might not persist across redeployments.

9.1.2. Verify the emulator thread pinning

Procedure

1. Identify the host and name for a given instance.

```bash
openstack server show <instance_id>
```
2. Use SSH to log on to the identified host as tripleo-admin.

```
ssh tripleo-admin@compute-1
[compute-1]$ sudo virsh dumpxml instance-00001 | grep "`emulatorpin cpuset"
```

### 9.2. TRUSTED VIRTUAL FUNCTIONS

You can configure trust between physical functions (PFs) and virtual functions (VFs), so that VFs can perform privileged actions, such as enabling promiscuous mode, or modifying a hardware address.

#### 9.2.1. Configuring trust between virtual and physical functions

**Prerequisites**

- An operational installation of Red Hat OpenStack Platform including director

**Procedure**

Complete the following steps to configure and deploy the overcloud with trust between physical and virtual functions:

1. Add the `NeutronPhysicalDevMappings` parameter in the `parameter_defaults` section to link between the logical network name and the physical interface.

   ```python
   parameter_defaults:
   NeutronPhysicalDevMappings:
   - sriov2:p5p2
   ``

2. Add the new property, `trusted`, to the SR-IOV parameters.

   ```python
   parameter_defaults:
   NeutronPhysicalDevMappings:
   - sriov2:p5p2
   NovaPCIPassthrough:
   - vendor_id: "8086"
   physical_network: "sriov2"
   trusted: "true"
   ``

   **NOTE**
   You must include double quotation marks around the value "true".

#### 9.2.2. Utilizing trusted VF networks

1. Create a network of type `vlan`.

   ```bash
   openstack network create trusted_vf_network --provider-network-type vlan \
   --provider-segment 111 --provider-physical-network sriov2 \
   --external --disable-port-security
   ```

2. Create a subnet.
openstack subnet create --network trusted_vf_network
  --ip-version 4 --subnet-range 192.168.111.0/24 --no-dhcp 
  subnet-trusted_vf_network

3. Create a port. Set the **vnic-type** option to **direct**, and the **binding-profile** option to **true**.

openstack port create --network sriov111
  --vnic-type direct --binding-profile trusted=true 
  sriov111_port_trusted

4. Create an instance, and bind it to the previously-created trusted port.

openstack server create --image rhel --flavor dpdk 
  --network internal --port trusted_vf_network_port_trusted --config-drive True --wait rhel-dpdk-sriov_trusted

**Verify the trusted VF configuration on the hypervisor**

1. On the compute node that you created the instance, enter the following command:

```bash
# ip link
7: p5p2: <BROADCAST,MULTICAST,UP,LOWER_UP> mtu 9000 qdisc mq state UP mode DEFAULT group default qlen 1000
  link/ether b4:96:91:1c:40:fa brd ff:ff:ff:ff:ff:ff
  vf 6 MAC fa:16:3e:b8:91:c2, vlan 111, spoof checking off, link-state auto, trust on, query_rss off
  vf 7 MAC fa:16:3e:84:cf:c8, vlan 111, spoof checking off, link-state auto, trust off, query_rss off
```

2. Verify that the trust status of the VF is **trust on**. The example output contains details of an environment that contains two ports. Note that **vf 6** contains the text **trust on**.

3. You can disable spoof checking if you set **port_security_enabled: false** in the Networking service (neutron) network, or if you include the argument **--disable-port-security** when you run the **openstack port create** command.

### 9.3. **CONFIGURING RX/TX QUEUE SIZE**

You can experience packet loss at high packet rates above 3.5 million packets per second (mpps) for many reasons, such as:

- a network interrupt
- a SMI
- packet processing latency in the Virtual Network Function

To prevent packet loss, increase the queue size from the default of 512 to a maximum of 1024.

**Prerequisites**

- To configure RX, ensure that you have libvirt v2.3 and QEMU v2.7.
- To configure TX, ensure that you have libvirt v3.7 and QEMU v2.10.
Procedure

- To increase the RX and TX queue size, include the following lines to the `parameter_defaults:` section of a relevant director role. Here is an example with ComputeOvsDpdk role:

```yaml
parameter_defaults:
  ComputeOvsDpdkParameters:
  - NovaLibvirtRxQueueSize: 1024
  - NovaLibvirtTxQueueSize: 1024
```

Testing

- You can observe the values for RX queue size and TX queue size in the nova.conf file:

```text
[libvirt]
rx_queue_size=1024
tx_queue_size=1024
```

- You can check the values for RX queue size and TX queue size in the VM instance XML file generated by libvirt on the compute host.

```xml
<devices>
  <interface type='vhostuser'>
    <mac address='56:48:4f:4d:5e:6f'/>
    <source type='unix' path='/tmp/vhost-user1' mode='server'/>
    <model type='virtio'/>
    <driver name='vhost' rx_queue_size='1024' tx_queue_size='1024'/>
    <address type='pci' domain='0x0000' bus='0x00' slot='0x10' function='0x0'/>
  </interface>
</devices>
```

To verify the values for RX queue size and TX queue size, use the following command on a KVM host:

```
$ virsh dumpxml <vm name> | grep queue_size
```

- You can check for improved performance, such as 3.8 mpps/core at 0 frame loss.

9.4. CONFIGURING A NUMA-AWARE VSWITCH

**IMPORTANT**

This feature is available in this release as a *Technology Preview*, and therefore is not fully supported by Red Hat. It should only be used for testing, and should not be deployed in a production environment. For more information about Technology Preview features, see Scope of Coverage Details.

Before you implement a NUMA-aware vSwitch, examine the following components of your hardware configuration:

- The number of physical networks.
- The placement of PCI cards.
Memory-mapped I/O (MMIO) devices, such as PCIe NICs, are associated with specific NUMA nodes. When a VM and the NIC are on different NUMA nodes, there is a significant decrease in performance. To increase performance, align PCIe NIC placement and instance processing on the same NUMA node.

Use this feature to ensure that instances that share a physical network are located on the same NUMA node. To optimize utilization of datacenter hardware, you must use multiple physnets.

**WARNING**

To configure NUMA-aware networks for optimal server utilization, you must understand the mapping of the PCIe slot and the NUMA node. For detailed information on your specific hardware, refer to your vendor’s documentation. If you fail to plan or implement your NUMA-aware vSwitch correctly, you can cause the servers to use only a single NUMA node.

To prevent a cross-NUMA configuration, place the VM on the correct NUMA node, by providing the location of the NIC to Nova.

**Prerequisites**

- You have enabled the filter `NUMATopologyFilter`

**Procedure**

- Set a new `NeutronPhysnetNUMANodesMapping` parameter to map the physical network to the NUMA node that you associate with the physical network.

- If you use tunnels, such as VxLAN or GRE, you must also set the `NeutronTunnelNUMANodes` parameter.

```plaintext
parameter_defaults:
    NeutronPhysnetNUMANodesMapping: {<physnet_name>: [<NUMA_NODE>]}
    NeutronTunnelNUMANodes: <NUMA_NODE>,<NUMA_NODE>
```

Here is an example with two physical networks tunneled to NUMA node 0:

- one project network associated with NUMA node 0
- one management network without any affinity

```plaintext
parameter_defaults:
    NeutronBridgeMappings:
      - tenant:br-link0
    NeutronPhysnetNUMANodesMapping: {tenant: [1], mgmt: [0,1]}
    NeutronTunnelNUMANodes: 0
```

- In the below example, assign the phsnet of the device named eno2 to NUMA number 0.

```plaintext
# ethtool -i eno2
```
Observe the physnet settings in the below example heat template.

NeutronBridgeMappings: 'physnet1:br-physnet1'
NeutronPhysnetNUMANodesMapping: {physnet1: [0] }

- type: ovs_user_bridge
  name: br-physnet1
  mtu: 9000
  members:
    - type: ovs_dpdk_port
      name: dpdk2
      members:
        - type: interface
          name: eno2

Testing NUMA-aware vSwitch

- Observe the configuration in the file /var/lib/config-data/puppet-generated/nova_libvirt/etc/nova/nova.conf

[neutron_physnet_tenant]
numa_nodes=1
[neutron_tunnel]
numa_nodes=1

- Confirm the new configuration with the `lscpu` command:

  $ lscpu

- Launch a VM, with the NIC attached to the appropriate network

Known Limitations

- You cannot start a VM that has two NICs connected to physnets on different NUMA nodes, if you did not specify a two-node guest NUMA topology.

- You cannot start a VM that has one NIC connected to a physnet and another NIC connected to a tunneled network on different NUMA nodes, if you did not specify a two-node guest NUMA topology.

- You cannot start a VM that has one vhost port and one VF on different NUMA nodes, if you did not specify a two-node guest NUMA topology.

- NUMA-aware vSwitch parameters are specific to overcloud roles. For example, Compute node 1 and Compute node 2 can have different NUMA topologies.

- If the interfaces of a VM have NUMA affinity, ensure that the affinity is for a single NUMA node only. You can locate any interface without NUMA affinity on any NUMA node.

- Configure NUMA affinity for data plane networks, not management networks.
9.5. CONFIGURING QUALITY OF SERVICE (QOS) IN AN NFVI ENVIRONMENT

For details on configuring QoS, see Configuring Quality of Service (QoS) policies. Support is limited to the following QoS rule types:

- **minimum bandwidth** on SR-IOV, if supported by vendor.
- **bandwidth limit** on SR-IOV and OVS-DPDK egress interfaces.

9.6. DEPLOYING AN OVERCLOUD WITH HCI AND DPDK

You can deploy your NFV infrastructure with hyper-converged nodes, by co-locating and configuring Compute and Ceph Storage services for optimized resource usage.

For more information about hyper-converged infrastructure (HCI), see the Deploying a hyperconverged infrastructure guide.

Prerequisites

- Red Hat OpenStack Platform (RHOSP) 17.1 or later.
- The latest version of Red Hat Ceph Storage 6.1.

Procedure

1. Generate the roles_data.yaml file for the Controller and the ComputeHCIOvsDpdk roles.

   ```
   $ openstack overcloud roles generate -o ~/<templates>/roles_data.yaml \
   Controller ComputeHCIOvsDpdk
   ```

2. Create and configure a new flavor with the openstack flavor create and openstack flavor set commands.
   For more information, see Composable services and custom roles in the Installing and managing Red Hat OpenStack Platform with director guide.

3. Deploy Ceph using RHOSP director.
   For more information, see Configuring the Red Hat Ceph Storage cluster in Deploying Red Hat Ceph Storage and Red Hat OpenStack Platform together with director.

4. Deploy the overcloud with the custom roles_data.yaml file that you generated.

   ```
   $ openstack overcloud deploy --templates \
   --timeout 360 \ 
   -r ~/<templates>/roles_data.yaml \
   -e /usr/share/openstack-tripleo-heat-templates/environments/\ 
   cephadm/cephadm-rbd-only.yaml \ 
   -e /usr/share/openstack-tripleo-heat-templates/environments/network-isolation.yaml \
   ```
IMPORTANT

This example deploys Ceph RBD (block storage) without Ceph RGW (object storage). To include RGW in the deployment, use `cephadm.yaml` instead of `cephadm-rbd-only.yaml`.

For more information, see *Deploying Red Hat Ceph Storage and Red Hat OpenStack Platform together with director.*

9.6.1. Example NUMA node configuration

For increased performance, place the tenant network and Ceph object service daemons (OSDs) in one NUMA node, such as NUMA-0, and the VNF and any non-NFV VMs in another NUMA node, such as NUMA-1.

**CPU allocation:**

<table>
<thead>
<tr>
<th>NUMA-0</th>
<th>NUMA-1</th>
</tr>
</thead>
<tbody>
<tr>
<td>Number of Ceph OSDs * 4 HT</td>
<td>Guest vCPU for the VNF and non-NFV VMs</td>
</tr>
<tr>
<td>DPDK lcore - 2 HT</td>
<td>DPDK lcore - 2 HT</td>
</tr>
<tr>
<td>DPDK PMD - 2 HT</td>
<td>DPDK PMD - 2 HT</td>
</tr>
</tbody>
</table>

**Example of CPU allocation:**

<table>
<thead>
<tr>
<th></th>
<th>NUMA-0</th>
<th>NUMA-1</th>
</tr>
</thead>
<tbody>
<tr>
<td>Ceph OSD</td>
<td>32,34,36,38,40,42,76,78,80,82,84,86</td>
<td></td>
</tr>
<tr>
<td>DPDK-lcore</td>
<td>0,44</td>
<td>1,45</td>
</tr>
<tr>
<td>DPDK-pmd</td>
<td>2,46</td>
<td>3,47</td>
</tr>
</tbody>
</table>

9.6.2. Example ceph configuration file

```
parameter_defaults:
  CephPoolDefaultSize: 3
  CephPoolDefaultPgNum: 64
  CephPools:
```
Assign CPU resources for ceph OSD processes with the following parameters. Adjust the values based on the workload and hardware in this hyperconverged environment.

1. **ceph_osd_docker_cpuset_cpus**: Allocate 4 CPU threads for each OSD for SSD disks, or 1 CPU for each OSD for HDD disks. Include the list of cores and sibling threads from the NUMA node associated with ceph, and the CPUs not found in the three lists: `NovaComputeCpuDedicatedSet`, and `OvsPmdCoreList`.

2. **ceph_osd_docker_cpu_limit**: Set this value to 0, to pin the ceph OSDs to the CPU list from `ceph_osd_docker_cpuset_cpus`.

3. **ceph_osd_numactl_opts**: Set this value to `preferred` for cross-NUMA operations, as a precaution.

### 9.6.3. Example DPDK configuration file

```bash
- {"name": backups, "pg_num": 128, "pgp_num": 128, "application": "rbd"}
- {"name": volumes, "pg_num": 256, "pgp_num": 256, "application": "rbd"}
- {"name": vms, "pg_num": 64, "pgp_num": 64, "application": "rbd"}
- {"name": images, "pg_num": 32, "pgp_num": 32, "application": "rbd"}

CephConfigOverrides:
  osd_recovery_op_priority: 3
  osd_recovery_max_active: 3
  osd_max_backfills: 1

CephAnsibleExtraConfig:
  nb_retry_wait_osd_up: 60
  delay_wait_osd_up: 20
  is_hci: true

# 3 OSDs * 4 vCPUs per SSD = 12 vCPUs (list below not used for VNF)
ceph_osd_docker_cpuset_cpus: "32,34,36,38,40,42,76,78,80,82,84,86" # 1
# cpu_limit 0 means no limit as we are limiting CPUs with cpuset above
ceph_osd_docker_cpu_limit: 0 # 2
# numactl preferred to cross the numa boundary if we have to
# but try to only use memory from numa node0
# cpuset-mems would not let it cross numa boundary
# lots of memory so NUMA boundary crossing unlikely
ceph_osd_numactl_opts: "-N 0 --preferred=0" # 3

CephAnsible_disks_config:
  osds_per_device: 1
  osd_scenario: lvm
  osd_object_store: bluestore
  devices:
    - /dev/sda
    - /dev/sdb
    - /dev/sdc

Assign CPU resources for ceph OSD processes with the following parameters. Adjust the values based on the workload and hardware in this hyperconverged environment.

1. **ceph_osd_docker_cpuset_cpus**: Allocate 4 CPU threads for each OSD for SSD disks, or 1 CPU for each OSD for HDD disks. Include the list of cores and sibling threads from the NUMA node associated with ceph, and the CPUs not found in the three lists.

2. **ceph_osd_docker_cpu_limit**: Set this value to 0, to pin the ceph OSDs to the CPU list from `ceph_osd_docker_cpuset_cpus`.

3. **ceph_osd_numactl_opts**: Set this value to `preferred` for cross-NUMA operations, as a precaution.

### 9.6.3. Example DPDK configuration file

```bash
parameter_defaults:
  ComputeHCIParameters:
    KernelArgs: "default_hugepagesz=1GB hugepagesz=1G hugepages=240 intel_iommu=on
    isolcpus=2,4,6,8,10,12,14,16,18,20,22,24,26,28,30,32,34,36,38,40,42,44,46,48,50,52,54,56,58,60,62,64,66,68,70,72,74,76,78,80,82,84,86" # 1
    TunedProfileName: "cpu-partitioning"
    IsolCpusList: # 2
```
1 KernelArgs: To calculate hugepages, subtract the value of the NovaReservedHostMemory parameter from total memory.

2 IsolCpusList: Assign a set of CPU cores that you want to isolate from the host processes with this parameter. Add the value of the OvsPmdCoreList parameter to the value of the NovaComputeCpuDedicatedSet parameter to calculate the value for the IsolCpusList parameter.

3 OvsDpdkSocketMemory: Specify the amount of memory in MB to pre-allocate from the hugepage pool per NUMA node with the OvsDpdkSocketMemory parameter. For more information about calculating OVS-DPDK parameters, see: ovsdpdk parameters

4 OvsPmdCoreList: Specify the CPU cores that are used for the DPDK poll mode drivers (PMD) with this parameter. Choose CPU cores that are associated with the local NUMA nodes of the DPDK interfaces. Allocate 2 HT sibling threads for each NUMA node to calculate the value for the OvsPmdCoreList parameter.

9.6.4. Example nova configuration file

```
parameter_defaults:
  ComputeHCIExtraConfig:
    nova::cpu_allocation_ratio: 16 # 2

NovaReservedHugePages: # 1
  - node:0,size:1GB,count:4
  - node:1,size:1GB,count:4

NovaReservedHostMemory: 123904 # All left over cpus from NUMA-1

NovaComputeCpuDedicatedSet: # 3

NovaReservedHugePages: Pre-allocate memory in MB from the hugepage pool with the NovaReservedHugePages parameter. It is the same memory total as the value for the OvsDpdkSocketMemory parameter.

NovaReservedHostMemory: Reserve memory in MB for tasks on the host with the NovaReservedHostMemory parameter. Use the following guidelines to calculate the amount of memory that you must reserve:

- 5 GB for each OSD.
- 0.5 GB overhead for each VM.
- 4GB for general host processing. Ensure that you allocate sufficient memory to prevent potential performance degradation caused by cross-NUMA OSD operation.
```
NovaComputeCpuDedicatedSet: List the CPUs not found in OvsPmdCoreList, or Ceph_osd_docker_cpuset_cpus with the NovaComputeCpuDedicatedSet parameter. The

9.6.5. Recommended configuration for HCI-DPDK deployments

Table 9.1. Tunable parameters for HCI deployments

<table>
<thead>
<tr>
<th>Block Device Type</th>
<th>OSDs, Memory, vCPUs per device</th>
</tr>
</thead>
<tbody>
<tr>
<td>NVMe</td>
<td>Memory: 5GB per OSD</td>
</tr>
<tr>
<td></td>
<td>OSDs per device: 4</td>
</tr>
<tr>
<td></td>
<td>vCPUs per device: 3</td>
</tr>
<tr>
<td>SSD</td>
<td>Memory: 5GB per OSD</td>
</tr>
<tr>
<td></td>
<td>OSDs per device: 1</td>
</tr>
<tr>
<td></td>
<td>vCPUs per device: 4</td>
</tr>
<tr>
<td>HDD</td>
<td>Memory: 5GB per OSD</td>
</tr>
<tr>
<td></td>
<td>OSDs per device: 1</td>
</tr>
<tr>
<td></td>
<td>vCPUs per device: 1</td>
</tr>
</tbody>
</table>

Use the same NUMA node for the following functions:

- Disk controller
- Storage networks
- Storage CPU and memory

Allocate another NUMA node for the following functions of the DPDK provider network:

- NIC
- PMD CPUs
- Socket memory

9.7. SYNCHRONIZE YOUR COMPUTE NODES WITH TIMEMASTER

IMPORTANT

This feature is available in this release as a Technology Preview, and therefore is not fully supported by Red Hat. It should only be used for testing, and should not be deployed in a production environment. For more information about Technology Preview features, see Scope of Coverage Details.

Use time protocols to maintain a consistent timestamp between systems.

Red Hat OpenStack Platform (RHOSP) includes support for Precision Time Protocol (PTP) and Network Time Protocol (NTP). You can use NTP to synchronize clocks in your network in the millisecond range, and you can use PTP to synchronize clocks to a higher, sub-microsecond, accuracy. An example
use case for PTP is a virtual radio access network (vRAN) that contains multiple antennas which provide higher throughput with more risk of interference.

Timemaster is a program that uses ptp4l and phc2sys in combination with chronyd or ntpd to synchronize the system clock to NTP and PTP time sources. The phc2sys and ptp4l programs use Shared Memory Driver (SHM) reference clocks to send PTP time to chronyd or ntpd, which compares the time sources to synchronize the system clock.

The implementation of the PTPv2 protocol in the Red Hat Enterprise Linux (RHEL) kernel is linuxptp. The linuxptp package includes the ptp4l program for PTP boundary clock and ordinary clock synchronization, and the phc2sys program for hardware time stamping. For more information about PTP, see: Introduction to PTP in the Red Hat Enterprise Linux System Administrator’s Guide.

Chrony is an implementation of the NTP protocol. The two main components of Chrony are chronyd, which is the Chrony daemon, and chronyc which is the Chrony command line interface. For more information about Chrony, see: Using chrony to configure ntp in the Red Hat Enterprise Linux System Administrator’s Guide.

The following image is an overview of a packet journey in a PTP configuration.

[Image of packet journey]

The following image is a overview of a packet journey in the Compute node in a PTP configuration.

[Image of packet journey]
9.7.1. Timemaster hardware requirements

Ensure that you have the following hardware functionality:

- You have configured the NICs with hardware timestamping capability.
- You have configured the switch to allow multicast packets.
- You have configured the switch to also function as a boundary or transparent clock.

You can verify the hardware timestamping with the command `ethtool -T <device>`.  

```
$ ethtool -T p5p1
Time stamping parameters for p5p1:
Capabilities:
  hardware-transmit    (SOF_TIMESTAMPING_TX_HARDWARE)
  software-transmit    (SOF_TIMESTAMPING_TX_SOFTWARE)
  hardware-receive     (SOF_TIMESTAMPING_RX_HARDWARE)
  software-receive     (SOF_TIMESTAMPING_RX_SOFTWARE)
  software-system-clock (SOF_TIMESTAMPING_SOFTWARE)
  hardware-raw-clock   (SOF_TIMESTAMPING_RAW_HARDWARE)
PTP Hardware Clock: 6
Hardware Transmit Timestamp Modes:
  off               (HWTSTAMP_TX_OFF)
  on                (HWTSTAMP_TX_ON)
Hardware Receive Filter Modes:
  none             (HWTSTAMP_FILTER_NONE)
  ptpv1-l4-sync    (HWTSTAMP_FILTER_PTP_V1_L4_SYNC)
  ptpv1-l4-delay-req (HWTSTAMP_FILTER_PTP_V1_L4_DELAY_REQ)
  ptpv2-event      (HWTSTAMP_FILTER_PTP_V2_EVENT)
```

You can use either a transparent or boundary clock switch for better accuracy and less latency. You can use an uplink switch for the boundary clock. The boundary clock switch uses an 8-bit `correctionField` on the PTPv2 header to correct delay variations, and ensure greater accuracy on the end clock. In a transparent clock switch, the end clock calculates the delay variation, not the `correctionField`.  

Red Hat OpenStack Platform 17.1-Beta Configuring network functions virtualization
9.7.2. Configuring Timemaster

The default Red Hat OpenStack Platform (RHOSP) service for time synchronization in overcloud nodes is **OS::TripleO::Services::Timesync**.

**Known limitations**

- Enable NTP for virtualized controllers, and enable PTP for bare metal nodes.
- Virtio interfaces are incompatible, because **ptp4l** requires a compatible PTP device.
- Use a physical function (PF) for a VM with SR-IOV. A virtual function (VF) does not expose the registers necessary for PTP, and a VM uses **kvm_ptp** to calculate time.
- High Availability (HA) interfaces with multiple sources and multiple network paths are incompatible.

**Procedure**

1. To enable the Timemaster service on the nodes that belong to a role that you choose, replace the line that contains **OS::TripleO::Services::Timesync** with the line **OS::TripleO::Services::TimeMaster** in the **roles_data.yaml** file section for that role.

```yaml
#- OS::TripleO::Services::Timesync
- OS::TripleO::Services::TimeMaster
```

2. Configure the heat parameters for the compute role that you use.

```yaml
#Example
ComputeSriovParameters:
  PTPInterfaces: ‘0:eno1,1:eno2’
  PTPMessageTransport: ‘UDPv4’
```

3. Include the new environment file in the **openstack overcloud deploy** command with any other environment files that are relevant to your environment:

```bash
$ openstack overcloud deploy \
   --templates \n   ... \
   -e <existing_overcloud_environment_files> \n   -e <new_environment_file1> \n   -e <new_environment_file2> \n   ... \
```

- Replace `<existing_overcloud_environment_files>` with the list of environment files that are part of your existing deployment.
- Replace `<new_environment_file>` with the new environment file or files that you want to include in the overcloud deployment process.

**Verification**

- Use the command **phc_ctl**, installed with **ptp4linux**, to query the NIC hardware clock.
# phc_ctl <clock_name> get
# phc_ctl <clock_name> cmp

## 9.7.3. Example timemaster configuration

$ cat /etc/timemaster.conf
# Configuration file for timemaster

# [ntp_server ntp-server.local]
# minpoll 4
# maxpoll 4

[ptp_domain 0]
interfaces eno1
# ptp4l_setting network_transport l2
# delay 10e-6

[timemaster]
ntp_program chronyd

[chrony.conf]
#include /etc/chrony.conf
server clock.redhat.com iburst minpoll 6 maxpoll 10

[ntp.conf]
includefile /etc/ntp.conf

[ptp4l.conf]
#includefile /etc/ptp4l.conf
network_transport L2

[chronyd]
path /usr/sbin/chronyd

[ntpd]
path /usr/sbin/ntpd
options -u ntp:ntp -g

[phc2sys]
path /usr/sbin/phc2sys
#options -w

[ptp4l]
path /usr/sbin/ptp4l
#options -2 -i eno1

## 9.7.4. Example timemaster operation

$ systemctl status timemaster
●
  timemaster.service - Synchronize system clock to NTP and PTP time sources
      Loaded: loaded (/usr/lib/systemd/system/timemaster.service; enabled; vendor preset: disabled)
     Active: active (running) since Tue 2020-08-25 19:10:18 UTC; 2min 6s ago
   Main PID: 2573 (timemaster)
      Tasks: 6 (limit: 357097)
Memory: 5.1M
CGroup: /system.slice/timemaster.service
│   └─ 2573 /usr/sbin/timemaster -f /etc/timemaster.conf
│   └─ 2577 /usr/sbin/chronyd -n -f /var/run/timemaster/chrony.conf
│   └─ 2582 /usr/sbin/ptp4l -l 5 -f /var/run/timemaster/ptp4l.0.conf -H -i eno1
│   └─ 2583 /usr/sbin/phc2sys -l 5 -a -r -R 1.00 -z /var/run/timemaster/ptp4l.0.socket -t [0:eno1] -n
   0 -E ntpshm -M 0
│   └─ 2587 /usr/sbin/ptp4l -l 5 -f /var/run/timemaster/ptp4l.1.conf -H -i eno2
│   └─ 2588 /usr/sbin/phc2sys -l 5 -a -r -R 1.00 -z /var/run/timemaster/ptp4l.1.socket -t [0:eno2] -n
   0 -E ntpshm -M 1

Aug 25 19:11:53 computesriov-0 ptp4l[2587]: [152.562] [0:eno2] selected local clock
e4434b.fffe.4a0c24 as best master
CHAPTER 10. ENABLING RT-KVM FOR NFV WORKLOADS

To facilitate installing and configuring Red Hat Enterprise Linux Real Time KVM (RT-KVM), Red Hat OpenStack Platform provides the following features:

- A real-time Compute node role that provisions Red Hat Enterprise Linux for real-time.
- The additional RT-KVM kernel module.
- Automatic configuration of the Compute node.

10.1. PLANNING FOR YOUR RT-KVM COMPUTE NODES

You must use Red Hat certified servers for your RT-KVM Compute nodes. For more information, see: Red Hat Enterprise Linux for Real Time certified servers.

For details on how to enable the rhel-9-server-nfv-rpms repository for RT-KVM, and ensuring your system is up to date, see: Registering and updating your undercloud.

NOTE

You need a separate subscription to a Red Hat OpenStack Platform for Real Time SKU before you can access this repository.

Building the real-time image

1. Install the libguestfs-tools package on the undercloud to get the virt-customize tool:

   (undercloud) [stack@undercloud-0 ~]$ sudo dnf install libguestfs-tools

   IMPORTANT

   If you install the libguestfs-tools package on the undercloud, disable iscsid.socket to avoid port conflicts with the tripleo_iscsid service on the undercloud:

   $ sudo systemctl disable --now iscsid.socket

2. Extract the images:

   (undercloud) [stack@undercloud-0 ~]$ tar -xf /usr/share/rhosp-director-images/overcloud-hardened-uefi-full-17.0.x86_64.tar
   (undercloud) [stack@undercloud-0 ~]$ tar -xf /usr/share/rhosp-director-images/ironic-python-agent-17.0.x86_64.tar

3. Copy the default image:

   (undercloud) [stack@undercloud-0 ~]$ cp overcloud-hardened-uefi-full.qcow2 overcloud-realtime-compute.qcow2

4. Register your image to enable Red Hat repositories relevant to your customizations. Replace [username] and [password] with valid credentials in the following example.
virt-customize -a overcloud-realtime-compute.qcow2 --run-command \
'subscription-manager register --username=[username] --password=[password]' \
subscription-manager release --set 9.0

NOTE

For security, you can remove credentials from the history file if they are used on the command prompt. You can delete individual lines in history using the `history -d` command followed by the line number.

5. Find a list of pool IDs from your account’s subscriptions, and attach the appropriate pool ID to your image.

```bash
sudo subscription-manager list --all --available | less
... 
virt-customize -a overcloud-realtime-compute.qcow2 --run-command \
'subscription-manager attach --pool [pool-ID]' 
```

6. Add the repositories necessary for Red Hat OpenStack Platform with NFV.

```bash
virt-customize -a overcloud-realtime-compute.qcow2 --run-command \
'sudo subscription-manager repos --enable=rhel-8-for-x86_64-baseos-eus-rpms \ 
--enable=rhel-8-for-x86_64-appstream-eus-rpms \ 
--enable=rhel-8-for-x86_64-highavailability-eus-rpms \ 
--enable=ansible-2.9-for-rhel-8-x86_64-rpms \ 
--enable=openstack-beta-for-rhel-8-x86_64-rpms \ 
--enable=rhel-8-for-x86_64-nfv-rpms \ 
--enable=fast-datapath-for-rhel-8-x86_64-rpms'
```

7. Create a script to configure real-time capabilities on the image.

```bash
(undercloud) [stack@undercloud-0 ~]$ cat <<'EOF' > rt.sh
#!/bin/bash
set -eux

dnf -v -y --setopt=protected_packages= erase kernel.$(uname -m)
dnf -v -y install kernel-rt kernel-rt-kvm tuned-profiles-nfv-host
grubby --set-default /boot/vmlinuz*rt*
EOF
```

8. Run the script to configure the real-time image:

```bash
(undercloud) [stack@undercloud-0 ~]$ virt-customize -a overcloud-realtime-compute.qcow2 -v --run rt.sh 2>&1 | tee virt-customize.log
```

NOTE

If you see the following line in the `rt.sh` script output, "grubby fatal error: unable to find a suitable template", you can ignore this error.
9. Examine the `virt-customize.log` file that resulted from the previous command, to check that the packages installed correctly using the `rt.sh` script.

```
(undercloud) [stack@undercloud-0 ~]$ cat virt-customize.log | grep Verifying
Verifying : kernel-3.10.0-957.el7.x86_64                                 1/1
Verifying : 10 qemu-kvm-tools-rhev-2.12.0-18.el7_6.1.x86_64              1/8
Verifying : tuned-profiles-realtime-2.10.0-6.el7_6.3.noarch             2/8
Verifying : linux-firmware-20180911-69.git85c5d90.el7.noarch            3/8
Verifying : tuned-profiles-nfv-host-2.10.0-6.el7_6.3.noarch            4/8
Verifying : kernel-rt-kvm-3.10.0-957.10.1.rt56.921.el7.x86_64          5/8
Verifying : tuna-0.13-6.el7.noarch                                    6/8
Verifying : kernel-rt-3.10.0-957.10.1.rt56.921.el7.x86_64             7/8
Verifying : rt-setup-2.0-6.el7.x86_64                                  8/8
```

10. Relabel SELinux:

```
(undercloud) [stack@undercloud-0 ~]$ virt-customize -a overcloud-realtime-compute.qcow2 -selinux-relabel
```

11. Extract vmlinuz and initrd:

```
(undercloud) [stack@undercloud-0 ~]$ mkdir image
(undercloud) [stack@undercloud-0 ~]$ guestmount -a overcloud-realtime-compute.qcow2 -i -ro image
(undercloud) [stack@undercloud-0 ~]$ cp image/boot/vmlinuz-3.10.0-862.rt56.804.el7.x86_64 ./overcloud-realtime-compute.vmlinuz
(undercloud) [stack@undercloud-0 ~]$ cp image/boot/initramfs-3.10.0-862.rt56.804.el7.x86_64.img ./overcloud-realtime-compute.initrd
(undercloud) [stack@undercloud-0 ~]$ guestunmount image
```

**NOTE**

The software version in the `vmlinuz` and `initramfs` filenames vary with the kernel version.

12. Upload the image:

```
(undercloud) [stack@undercloud-0 ~]$ openstack overcloud image upload --update-existing -os-image-name overcloud-realtime-compute.qcow2
```

You now have a real-time image you can use with the `ComputeOvsDpdkRT` composable role on your selected Compute nodes.

**Modifying BIOS settings on RT-KVM Compute nodes**

To reduce latency on your RT-KVM Compute nodes, disable all options for the following parameters in your Compute node BIOS settings:

- Power Management
- Hyper-Threading
- CPU sleep states
Logical processors

See Setting BIOS parameters for descriptions of these settings and the impact of disabling them. See your hardware manufacturer documentation for complete details on how to change BIOS settings.

10.2. CONFIGURING OVS-DPDK WITH RT-KVM

10.2.1. Designating nodes for Real-time Compute

To designate nodes for Real-time Compute, create a new role file to configure the Real-time Compute role, and configure the bare-metal nodes with a Real-time Compute resource class to tag the Compute nodes for real-time.

NOTE

The following procedure applies to new overcloud nodes that you have not yet provisioned. To assign a resource class to an existing overcloud node that has already been provisioned, scale down the overcloud to unprovision the node, then scale up the overcloud to reprovision the node with the new resource class assignment. For more information, see Scaling overcloud nodes.

Procedure

1. Log in to the undercloud host as the stack user.

2. Source the stackrc undercloud credentials file:

   [stack@director ~]$ source ~/stackrc

3. Based on the /usr/share/openstack-tripleo-heat-templates/environments/compute-real-time-example.yaml file, create a compute-real-time.yaml environment file that sets the parameters for the ComputeRealTime role.

4. Generate a new roles data file named roles_data_rt.yaml that includes the ComputeRealTime role, along with any other roles that you need for the overcloud. The following example generates the roles data file roles_data_rt.yaml, which includes the roles Controller, Compute, and ComputeRealTime:

   (undercloud)$ openstack overcloud roles generate \
   -o /home/stack/templates/roles_data_rt.yaml \
   ComputeRealTime ComputeController

5. Update the roles_data_rt.yaml file for the ComputeRealTime role:

   # Role: ComputeRealTime
   # 
   - name: ComputeRealTime
description: |
   - Real Time Compute Node role
   CountDefault: 1
   
   # Create external Neutron bridge
tags:
   - compute
- external_bridge
networks:
  InternalApi:
    subnet: internal_api_subnet
  Tenant:
    subnet: tenant_subnet
  Storage:
    subnet: storage_subnet
HostnameFormatDefault: '%stackname%-computert-%index%'
deprecated_nic_config_name: compute-rt.yaml

6. Register the ComputeRealTime nodes for the overcloud by adding them to your node definition template: node.json or node.yaml. For more information, see Registering nodes for the overcloud in the Installing and managing Red Hat OpenStack Platform with director guide.

7. Inspect the node hardware:

   (undercloud)$ openstack overcloud node introspect --all-manageable --provide

   For more information, see Creating an inventory of the bare-metal node hardware in the Installing and managing Red Hat OpenStack Platform with director guide.

8. Tag each bare-metal node that you want to designate for ComputeRealTime with a custom ComputeRealTime resource class:

   (undercloud)$ openstack baremetal node set \
   --resource-class baremetal.RTCOMPUTE <node>

   Replace <node> with the name or UUID of the bare-metal node.

9. Add the ComputeRealTime role to your node definition file, overcloud-baremetal-deploy.yaml, and define any predictive node placements, resource classes, network topologies, or other attributes that you want to assign to your nodes:

   - name: Controller
count: 3
...
- name: Compute
count: 3
...
- name: ComputeRealTime
count: 1
defaults:
  resource_class: baremetal.RTCOMPUTE
network_config:
  template: /home/stack/templates/nic-config/<role_topology_file>

   • Replace <role_topology_file> with the name of the topology file to use for the ComputeRealTime role, for example, myRoleTopology.j2. You can reuse an existing network topology or create a new custom network interface template for the role. For more information, see Custom network interface templates in the Installing and managing Red Hat OpenStack Platform with director guide. To use the default network definition settings, do not include network_config in the role definition.
For more information about the properties you can use to configure node attributes in your node definition file, see `Bare-metal node provisioning attributes`. For an example node definition file, see `Example node definition file`.

10. Create the following Ansible playbook to configure the kernel during the node provisioning, and save the playbook as `/home/stack/templates/fix_rt_kernel.yaml`:

```yaml
# RealTime KVM fix until BZ #2122949 is closed-
- name: Fix RT Kernel
  hosts: allovercloud
  any_errors_fatal: true
  gather_facts: false
  vars:
    reboot_wait_timeout: 900
  pre_tasks:
    - name: Wait for provisioned nodes to boot
      wait_for_connection:
        timeout: 600
        delay: 10
  tasks:
    - name: Fix bootloader entry
      become: true
      shell: |
        set -eux
        new_entry=$(grep saved_entry= /boot/grub2/grubenv | sed -e s/saved_entry=//)
        source /etc/default/grub
        sed -i "s/options.*options root=$GRUB_DEVICE ro $GRUB_CMDLINE_LINUX $GRUB_CMDLINE_LINUX_DEFAULT/" /boot/loader/entries/$(</etc/machine-id)$new_entry.conf
        cp -f /boot/grub2/grubenv /boot/efi/EFI/redhat/grubenv
  post_tasks:
    - name: Configure reboot after new kernel
      become: true
      reboot:
        reboot_timeout: "{{ reboot_wait_timeout }}"
        when: reboot_wait_timeout is defined
```

11. Include `/home/stack/templates/fix_rt_kernel.yaml` as a playbook in the `ComputeOvsDpdkSriovRT` role definition in your node provisioning file:

```yaml
- name: ComputeOvsDpdkSriovRT
  ...  
  ansible_playbooks:
    - playbook: /usr/share/ansible/tripleo-playbooks/cli-overcloud-node-kernelargs.yaml
      extra_vars:
        kernel_args: "default_hugepagesz=1GB hugepagesz=1G hugepages=64 iommu=pt intel_iommu=on tsx=off isolcpus=2-19,22-39"
        reboot_wait_timeout: 900
        tuned_profile: "cpu-partitioning"
        tuned_isolated_cores: "2-19,22-39"
        defer_reboot: true
      - playbook: /home/stack/templates/fix_rt_kernel.yaml
        extra_vars:
          reboot_wait_timeout: 1800
```

CHAPTER 10. ENABLING RT-KVM FOR NFV WORKLOADS
For more information about the properties you can use to configure node attributes in your node definition file, see Bare-metal node provisioning attributes. For an example node definition file, see Example node definition file.

12. Provision the new nodes for your role:

```
(undercloud)$ openstack overcloud node provision \\
[--stack <stack> \ ] \\
[--network-config \] \\
--output <deployment_file> \ \\
/home/stack/templates/overcloud-baremetal-deploy.yaml
```

- Optional: Replace `<stack>` with the name of the stack for which the bare-metal nodes are provisioned. The default is `overcloud`.

- Optional: Include the `--network-config` optional argument to provide the network definitions to the `cli-overcloud-node-network-config.yaml` Ansible playbook. If you do not define the network definitions by using the `network_config` property, then the default network definitions are used.

- Replace `<deployment_file>` with the name of the heat environment file to generate for inclusion in the deployment command, for example `/home/stack/templates/overcloud-baremetal-deployed.yaml`.

13. Monitor the provisioning progress in a separate terminal. When provisioning is successful, the node state changes from `available` to `active`:

```
(undercloud)$ watch openstack baremetal node list
```

14. If you ran the provisioning command without the `--network-config` option, then configure the `<Role>NetworkConfigTemplate` parameters in your `network-environment.yaml` file to point to your NIC template files:

```
parameter_defaults:
  ComputeNetworkConfigTemplate: /home/stack/templates/nic-configs/compute.j2
  ComputeAMDSEVNetworkConfigTemplate: /home/stack/templates/nic-configs/<rt_compute>.j2
  ControllerNetworkConfigTemplate: /home/stack/templates/nic-configs/controller.j2
```

Replace `<rt_compute>` with the name of the file that contains the network topology of the `ComputeRealTime` role, for example, `computert.yaml` to use the default network topology.

15. Add your environment file to the stack with your other environment files and deploy the overcloud:

```
(undercloud)$ openstack overcloud deploy --templates \\
- r /home/stack/templates/roles_data_rt.yaml \\
- e /home/stack/templates/overcloud-baremetal-deployed.yaml \\
- e /home/stack/templates/node-info.yaml \\
- e [your environment files] \\
- e /home/stack/templates/compute-real-time.yaml
```

10.2.2. Configuring the OVS-DPDK parameters

1. Add the NIC configuration for the OVS-DPDK role you use under `resource_registry`:
2. Under `parameter_defaults`, set the OVS-DPDK, and RT-KVM parameters:

```bash
# DPDK compute node.
ComputeOvsDpdkRTParameters:
  KernelArgs: "default_hugepagesz=1GB hugepagesz=1G hugepages=32 iommu=pt
              intel_iommu=on isolcpus=1-7,17-23,9-15,25-31"
  TunedProfileName: "realtime-virtual-host"
  IsolCpusList:
    "1,2,3,4,5,6,7,9,10,17,18,19,20,21,22,23,11,12,13,14,15,26,27,28,29,30,31"
  NovaComputeCpuDedicatedSet: [2,3,4,5,6,7,18,19,20,21,22,23,10,11,12,13,14,15,26,27,28,29,30,31]
  NovaReservedHostMemory: 4096
  OvsDpdkSocketMemory: "1024,1024"
  OvsDpdkMemoryChannels: "4"
  OvsPmdCoreList: "1,17,9,25"
  VhostuserSocketGroup: "hugetlbfs"
  ComputeOvsDpdkRTImage: "overcloud-realtime-compute"
```

### 10.3. LAUNCHING AN RT-KVM INSTANCE

Perform the following steps to launch an RT-KVM instance on a real-time enabled Compute node:

1. Create an RT-KVM flavor on the overcloud:

   ```bash
   # openstack flavor create  r1.small 99 4096 20 4
   # openstack flavor set --property hw:cpu_policy=dedicated 99
   # openstack flavor set --property hw:cpu_realtime=yes 99
   # openstack flavor set --property hw:mem_page_size=1GB 99
   # openstack flavor set --property hw:cpu_realtime_mask="^0-1" 99
   # openstack flavor set --property hw:cpu_emulator_threads=isolate 99
   ```

2. Launch an RT-KVM instance:

   ```bash
   # openstack server create  --image <rhel> --flavor r1.small --nic net-id=<dpdk-net> test-rt
   ```

3. To verify that the instance uses the assigned emulator threads, run the following command:

   ```bash
   # virsh dumpxml <instance-id> | grep vcpu -A1
   <vcpu placement='static'>4</vcpu>
   <cputune>
     <vcpupin vcpu='0' cpuset='1'/>
     <vcpupin vcpu='1' cpuset='3'/>
     <vcpupin vcpu='2' cpuset='5'/>
     <vcpupin vcpu='3' cpuset='7'/>
     <emulatorpin cpuset='0-1'/>
   ```
<vcpusched vcpus='2-3' scheduler='lifo' priority='1'/>
</cputune>
CHAPTER 11. EXAMPLE: CONFIGURING OVS-DPDK AND SR-IOV WITH VXLAN TUNNELLING

You can deploy Compute nodes with both OVS-DPDK and SR-IOV interfaces. The cluster includes ML2/OVS and VXLAN tunnelling.

IMPORTANT

In your roles configuration file, for example `roles_data.yaml`, comment out or remove the line that contains `OS::TripleO::Services::Tuned`, when you generate the overcloud roles.

When you have commented out or removed `OS::TripleO::Services::Tuned`, you can set the `TunedProfileName` parameter to suit your requirements, for example "cpu-partitioning". If you do not comment out or remove the line `OS::TripleO::Services::Tuned` and redeploy, the `TunedProfileName` parameter gets the default value of "throughput-performance", instead of any other value that you set.

11.1. CONFIGURING ROLES DATA

Red Hat OpenStack Platform provides a set of default roles in the `roles_data.yaml` file. You can create your own `roles_data.yaml` file to support the roles you require.

For the purposes of this example, the ComputeOvsDpdkSriov role is created. For information on creating roles in Red Hat OpenStack Platform, see Advanced Overcloud Customization. For details on the specific role used for this example, see `roles_data.yaml`.

11.2. CONFIGURING OVS-DPDK PARAMETERS

1. Add the custom resources for OVS-DPDK under `resource_registry`:

   ```yaml
   resource_registry:
     # Specify the relative/absolute path to the config files you want to use for override the default.
     OS::TripleO::ComputeOvsDpdkSriov::Net::SoftwareConfig: nic-configs/computeovsdpdksriov.yaml
     OS::TripleO::Controller::Net::SoftwareConfig: nic-configs/controller.yaml
   ``

2. Under `parameter_defaults`, set the tunnel type to `vxlan`, and the network type to `vxlan,vlan`:

   ```yaml
   NeutronTunnelTypes: 'vxlan'
   NeutronNetworkType: 'vxlan,vlan'
   ``

3. Under `parameters_defaults`, set the bridge mapping:

   ```yaml
   # The OVS logical->physical bridge mappings to use.
   NeutronBridgeMappings:
     - dpdk-mgmt:br-link0
   ```
4. Under **parameter_defaults**, set the role-specific parameters for the `ComputeOvsDpdkSriov` role:

```bash
###############################
# OVS DPDK configuration #
###############################
ComputeOvsDpdkSriovParameters:
  KernelArgs: "default_hugepagesz=1GB hugepagesz=1G hugepages=32 iommu=pt
              intel_iommu=on isolcpus=2-19,22-39"
  TunedProfileName: "cpu-partitioning"
  IsolCpusList: "2-19,22-39"
  NovaComputeCpuDedicatedSet: [4-19,24-39]
  NovaReservedHostMemory: 4096
  OvsDpdkSocketMemory: "3072,1024"
  OvsDpdkMemoryChannels: "4"
  OvsPmdCoreList: "2,22,3,23"
  NovaComputeCpuSharedSet: [0,20,1,21]
  NovaLibvirtRxQueueSize: 1024
  NovaLibvirtTxQueueSize: 1024
```

**NOTE**

To prevent failures during guest creation, assign at least one CPU with sibling thread on each NUMA node. In the example, the values for the `OvsPmdCoreList` parameter denote cores 2 and 22 from NUMA 0, and cores 3 and 23 from NUMA 1.

**NOTE**

These huge pages are consumed by the virtual machines, and also by OVS-DPDK using the `OvsDpdkSocketMemory` parameter as shown in this procedure. The number of huge pages available for the virtual machines is the `boot` parameter minus the `OvsDpdkSocketMemory`. You must also add `hw:mem_page_size=1GB` to the flavor you associate with the DPDK instance.

**NOTE**

`OvsDpdkMemoryChannels` is a required setting for this procedure. For optimum operation, ensure you deploy DPDK with appropriate parameters and values.

5. Configure the role-specific parameters for SR-IOV:

```json
NovaPCIPassthrough:
  - vendor_id: "8086"
    product_id: "1528"
    address: "0000:06:00.0"
    trusted: "true"
    physical_network: "sriov-1"
  - vendor_id: "8086"
    product_id: "1528"
```
11.3. CONFIGURING THE CONTROLLER NODE

1. Create the control-plane Linux bond for an isolated network.

```yaml
- type: linux_bond
  name: bond_api
  bonding_options: "mode=active-backup"
  use_dhcp: false
  dns_servers:
    get_param: DnsServers
  members:
    - type: interface
      name: nic2
      primary: true
```

2. Assign VLANs to this Linux bond.

```yaml
- type: vlan
  vlan_id:
    get_param: InternalApiNetworkVlanID
  device: bond_api
  addresses:
    - ip_netmask:
        get_param: InternalApiIpSubnet

- type: vlan
  vlan_id:
    get_param: StorageNetworkVlanID
  device: bond_api
  addresses:
    - ip_netmask:
        get_param: StorageIpSubnet

- type: vlan
  vlan_id:
    get_param: StorageMgmtNetworkVlanID
  device: bond_api
  addresses:
    - ip_netmask:
        get_param: StorageMgmtIpSubnet

- type: vlan
  vlan_id:
    get_param: ExternalNetworkVlanID
  device: bond_api
  addresses:
    - ip_netmask:
        get_param: ExternalIpSubnet
  routes:
```
3. Create the OVS bridge to access `neutron-dhcp-agent` and `neutron-metadata-agent` services.

```yaml
- type: ovs_bridge
  name: br-link0
  use_dhcp: false
  mtu: 9000
  members:
    - type: interface
      name: nic3
      mtu: 9000
    - type: vlan
      vlan_id:
        get_param: TenantNetworkVlanID
      mtu: 9000
      addresses:
        - ip_netmask:
            get_param: TenantIpSubnet

- type: linux_bond
  name: bond_api
  bonding_options: "mode=active-backup"
  use_dhcp: false
  dns_servers:
    get_param: DnsServers
  members:
    - type: interface
      name: nic3
      primary: true
    - type: interface
      name: nic4
```

### 11.4. CONFIGURING THE COMPUTE NODE FOR DPDK AND SR-IOV

Create the `computeovsdpdksriov.yaml` file from the default `compute.yaml` file, and make the following changes:

1. Create the control-plane Linux bond for an isolated network.

```yaml
- type: linux_bond
  name: bond_api
  bonding_options: "mode=active-backup"
  use_dhcp: false
  dns_servers:
    get_param: DnsServers
  members:
    - type: interface
      name: nic3
      primary: true
    - type: interface
      name: nic4
```

2. Assign VLANs to this Linux bond.

```yaml
- type: vlan
  vlan_id:
    get_param: InternalApiNetworkVlanID
  device: bond_api
  addresses:
    - ip_netmask:
        get_param: InternalApiIpSubnet
```

```yaml
- type: vlan
  vlan_id:
```
3. Set a bridge with a DPDK port to link to the controller.

```bash
- type: ovs_user_bridge
  name: br-link0
  use_dhcp: false
  ovs_extra:
    - str_replace:
      template: set port br-link0 tag=_VLAN_TAG_
      params:
        _VLAN_TAG_:
          get_param: TenantNetworkVlanID
  addresses:
    - ip_netmask:
      get_param: TenantIpSubnet
  members:
    - type: ovs_dpdk_bond
      name: dpdkbond0
      mtu: 9000
      rx_queue: 2
      members:
        - type: ovs_dpdk_port
          name: dpdk0
          members:
            - type: interface
              name: nic7
        - type: ovs_dpdk_port
          name: dpdk1
          members:
            - type: interface
              name: nic8
```

**NOTE**

To include multiple DPDK devices, repeat the `type` code section for each DPDK device that you want to add.

**NOTE**

When using OVS-DPDK, all bridges on the same Compute node must be of type `ovs_user_bridge`. Red Hat OpenStack Platform does not support both `ovs_bridge` and `ovs_user_bridge` located on the same node.

### 11.5. DEPLOYING THE OVERCLOUD

1. Run the `overcloud_deploy.sh` script:
CHAPTER 12. UPGRADING RED HAT OPENSTACK PLATFORM WITH NFV

For more information about upgrading Red Hat OpenStack Platform (RHOSP) with OVS-DPDK configured, see Preparing network functions virtualization (NFV) in the Framework for Upgrades (13 to 16.1) Guide.
CHAPTER 13. NFV PERFORMANCE

Red Hat OpenStack Platform director configures the Compute nodes to enforce resource partitioning and fine tuning to achieve line rate performance for the guest virtual network functions (VNFs). The key performance factors in the NFV use case are throughput, latency, and jitter.

You can enable high-performance packet switching between physical NICs and virtual machines using data plane development kit (DPDK) accelerated virtual machines. OVS 2.10 embeds support for DPDK and includes support for vhost-user multiqueue, allowing scalable performance. OVS-DPDK provides line-rate performance for guest VNFs.

Single root I/O virtualization (SR-IOV) networking provides enhanced performance, including improved throughput for specific networks and virtual machines.

Other important features for performance tuning include huge pages, NUMA alignment, host isolation, and CPU pinning. VNF flavors require huge pages and emulator thread isolation for better performance. Host isolation and CPU pinning improve NFV performance and prevent spurious packet loss.

For a high-level introduction to CPUs and NUMA topology, see: NFV Performance Considerations and Configuring emulator threads.
CHAPTER 14. FINDING MORE INFORMATION

The following table includes additional Red Hat documentation for reference:

The Red Hat OpenStack Platform documentation suite can be found here: Red Hat OpenStack Platform Documentation Suite

Table 14.1. List of Available Documentation

<table>
<thead>
<tr>
<th>Component</th>
<th>Reference</th>
</tr>
</thead>
<tbody>
<tr>
<td>Red Hat Enterprise Linux</td>
<td>Red Hat OpenStack Platform is supported on Red Hat Enterprise Linux 8.0. For information on installing Red Hat Enterprise Linux, see the corresponding installation guide at: Red Hat Enterprise Linux Documentation Suite.</td>
</tr>
<tr>
<td>Red Hat OpenStack Platform</td>
<td>To install OpenStack components and their dependencies, use the Red Hat OpenStack Platform director. The director uses a basic OpenStack installation as the undercloud to install, configure, and manage the OpenStack nodes in the final overcloud. Ensure that you have one extra host machine for the installation of the undercloud, in addition to the environment necessary for the deployed overcloud. For detailed instructions, see Red Hat OpenStack Platform Installing and managing Red Hat OpenStack Platform with director. For information on configuring advanced features for a Red Hat OpenStack Platform enterprise environment using the Red Hat OpenStack Platform director, such as network isolation, storage configuration, SSL communication, and general configuration method, see Advanced Overcloud Customization.</td>
</tr>
<tr>
<td>NFV Documentation</td>
<td>For more details on planning and configuring your Red Hat OpenStack Platform deployment with single root I/O virtualization (SR-IOV) and Open vSwitch with Data Plane Development Kit (OVS-DPDK), see Network Function Virtualization Planning and Configuration Guide.</td>
</tr>
</tbody>
</table>
CHAPTER 15. SAMPLE DPDK SRIOV YAML FILES

This section provides sample yaml files as a reference to add single root I/O virtualization (SR-IOV) and Data Plane Development Kit (DPDK) interfaces on the same compute node.

NOTE

These templates are from a fully-configured environment, and include parameters unrelated to NFV, that might not apply to your deployment. For a list of component support levels, see the Red Hat Knowledgebase solution Component Support Graduation.

15.1. SAMPLE VXLAN DPDK SRIOV YAML FILES

15.1.1. roles_data.yaml

1. Run the openstack overcloud roles generate command to generate the roles_data.yaml file. Include role names in the command according to the roles that you want to deploy in your environment, such as Controller, ComputeSriov, ComputeOvsDpdkRT, ComputeOvsDpdkSriov, or other roles. For example, to generate a roles_data.yaml file that contains the roles Controller and ComputeHCIovsDpdkSriov, run the following command:

```
$ openstack overcloud roles generate -o roles_data.yaml Controller ComputeHCIovsDpdkSriov
```

```
# File generated by TripleO
# # Role: Controller
# #
- name: Controller
description: |
  Controller role that has all the controller services loaded and handles Database, Messaging and Network functions.
CountDefault: 1
tags:
  - primary
  - controller
networks:
  External:
    subnet: external_subnet
  InternalApi:
    subnet: internal_api_subnet
  Storage:
    subnet: storage_subnet
  StorageMgmt:
    subnet: storage_mgmt_subnet
  Tenant:
    subnet: tenant_subnet
# For systems with both IPv4 and IPv6, you may specify a gateway network for each, such as ['ControlPlane', 'External']
default_route_networks: ['External']
HostnameFormatDefault: '%stackname%-controller-%index%'
```
# Deprecated & backward-compatible values (FIXME: Make parameters consistent)
# Set uses_deprecated_params to True if any deprecated params are used.
uses_deprecated_params: True
deprecated_param_extraconfig: 'controllerExtraConfig'
deprecated_param_flavor: 'OvercloudControlFlavor'
deprecated_param_image: 'controllerImage'
deprecated_nic_config_name: 'controller.yaml'
update_serial: 1

ServicesDefault:
- OS::TripleO::Services::Aide
- OS::TripleO::Services::AodhApi
- OS::TripleO::Services::AodhEvaluator
- OS::TripleO::Services::AodhListener
- OS::TripleO::Services::AodhNotifier
- OS::TripleO::Services::AuditD
- OS::TripleO::Services::BarbicanApi
- OS::TripleO::Services::BarbicanBackendSimpleCrypto
- OS::TripleO::Services::BarbicanBackendDogtag
- OS::TripleO::Services::BarbicanBackendKmip
- OS::TripleO::Services::BarbicanBackendPkcs11Crypto
- OS::TripleO::Services::BootParams
- OS::TripleO::Services::CACerts
- OS::TripleO::Services::CeilometerAgentCentral
- OS::TripleO::Services::CeilometerAgentNotification
- OS::TripleO::Services::CephExternal
- OS::TripleO::Services::CephGrafana
- OS::TripleO::Services::CephMds
- OS::TripleO::Services::CephMgr
- OS::TripleO::Services::CephMon
- OS::TripleO::Services::CephRbdMirror
- OS::TripleO::Services::CephRgw
- OS::TripleO::Services::CertmongerUser
- OS::TripleO::Services::CinderApi
- OS::TripleO::Services::CinderBackendDellPs
- OS::TripleO::Services::CinderBackendDellSc
- OS::TripleO::Services::CinderBackendDellEMCPowermax
- OS::TripleO::Services::CinderBackendDellEMCPowerStore
- OS::TripleO::Services::CinderBackendDellEMCSc
- OS::TripleO::Services::CinderBackendDellEMCUinity
- OS::TripleO::Services::CinderBackendDellEMCVMAXISCSI
- OS::TripleO::Services::CinderBackendDellEMCVNX
- OS::TripleO::Services::CinderBackendDellEMCVxFlexOS
- OS::TripleO::Services::CinderBackendDellEMCXtremio
- OS::TripleO::Services::CinderBackendDellEMCXTEMRIOSCS1
- OS::TripleO::Services::CinderBackendDellNetApp
- OS::TripleO::Services::CinderBackendPure
- OS::TripleO::Services::CinderBackendScaleIO
- OS::TripleO::Services::CinderBackendVRTSHyperScale
- OS::TripleO::Services::CinderBackendNVMeOF
- OS::TripleO::Services::CinderBackup
- OS::TripleO::Services::CinderHPELeftHandISCSI
- OS::TripleO::Services::CinderScheduler
- OS::TripleO::Services::CinderVolume
- OS::TripleO::Services::Clustercheck
- OS::TripleO::Services::Collectd
- OS::TripleO::Services::ContainerImagePrepare
CHAPTER 15. SAMPLE DPDK SRIOV YAML FILES

- OS::TripleO::Services::DesignateApi
- OS::TripleO::Services::DesignateCentral
- OS::TripleO::Services::DesignateProducer
- OS::TripleO::Services::DesignateWorker
- OS::TripleO::Services::DesignateMDNS
- OS::TripleO::Services::DesignateSink
- OS::TripleO::Services::Docker
- OS::TripleO::Services::Ec2Api
- OS::TripleO::Services::Etcd
- OS::TripleO::Services::ExternalSwiftProxy
- OS::TripleO::Services::GlanceApi
- OS::TripleO::Services::GnocchiApi
- OS::TripleO::Services::GnocchiMetricd
- OS::TripleO::Services::GnocchiStatsd
- OS::TripleO::Services::HAProxy
- OS::TripleO::Services::HeatApi
- OS::TripleO::Services::HeatApiCloudwatch
- OS::TripleO::Services::HeatApiCfn
- OS::TripleO::Services::HeatEngine
- OS::TripleO::Services::Horizon
- OS::TripleO::Services::IpaClient
- OS::TripleO::Services::Ipsec
- OS::TripleO::Services::IronicApi
- OS::TripleO::Services::IronicConductor
- OS::TripleO::Services::IronicInspector
- OS::TripleO::Services::IronicPxe
- OS::TripleO::Services::IronicNeutronAgent
- OS::TripleO::Services::Iscsid
- OS::TripleO::Services::Keepalived
- OS::TripleO::Services::Kernel
- OS::TripleO::Services::Keystone
- OS::TripleO::Services::LoginDefs
- OS::TripleO::Services::ManilaApi
- OS::TripleO::Services::ManilaBackendCephFs
- OS::TripleO::Services::ManilaBackendIsilon
- OS::TripleO::Services::ManilaBackendNetapp
- OS::TripleO::Services::ManilaBackendUnity
- OS::TripleO::Services::ManilaBackendVNX
- OS::TripleO::Services::ManilaBackendVMAX
- OS::TripleO::Services::ManilaScheduler
- OS::TripleO::Services::ManilaShare
- OS::TripleO::Services::Memcached
- OS::TripleO::Services::MetricsQdr
- OS::TripleO::Services::MistralApi
- OS::TripleO::Services::MistralEngine
- OS::TripleO::Services::MistralExecutor
- OS::TripleO::Services::MistralEventEngine
- OS::TripleO::Services::Multipathd
- OS::TripleO::Services::MySQL
- OS::TripleO::Services::MySQLClient
- OS::TripleO::Services::NeutronApi
- OS::TripleO::Services::NeutronBgpVpnApi
- OS::TripleO::Services::NeutronSfcApi
- OS::TripleO::Services::NeutronCorePlugin
- OS::TripleO::Services::NeutronDhcpAgent
- OS::TripleO::Services::NeutronL2gwAgent
- OS::TripleO::Services::NeutronL2gwApi
- OS::TripleO::Services::NeutronL3Agent
- OS::TripleO::Services::NeutronLinuxbridgeAgent
- OS::TripleO::Services::NeutronMetadataAgent
- OS::TripleO::Services::NeutronML2FujitsuCfab
- OS::TripleO::Services::NeutronML2FujitsuFossw
- OS::TripleO::Services::NeutronOvsAgent
- OS::TripleO::Services::NeutronVppAgent
- OS::TripleO::Services::NeutronAgentsIBConfig
- OS::TripleO::Services::NovaApi
- OS::TripleO::Services::NovaConductor
- OS::TripleO::Services::NovaIronic
- OS::TripleO::Services::NovaMetadata
- OS::TripleO::Services::NovaScheduler
- OS::TripleO::Services::NovaVncProxy
- OS::TripleO::Services::ContainersLogrotateCron
- OS::TripleO::Services::OctaviaApi
- OS::TripleO::Services::OctaviaDeploymentConfig
- OS::TripleO::Services::OctaviaHealthManager
- OS::TripleO::Services::OctaviaHousekeeping
- OS::TripleO::Services::OctaviaWorker
- OS::TripleO::Services::OpenStackClients
- OS::TripleO::Services::OVNDBs
- OS::TripleO::Services::OVNController
- OS::TripleO::Services::Pacemaker
- OS::TripleO::Services::PankoApi
- OS::TripleO::Services::PlacementApi
- OS::TripleO::Services::OsloMessagingRpc
- OS::TripleO::Services::OsloMessagingNotify
- OS::TripleO::Services::Podman
- OS::TripleO::Services::Rear
- OS::TripleO::Services::Redis
- OS::TripleO::Services::Rhsm
- OS::TripleO::Services::Rsyslog
- OS::TripleO::Services::RsyslogSidecar
- OS::TripleO::Services::SaharaApi
- OS::TripleO::Services::SaharaEngine
- OS::TripleO::Services::Securetty
- OS::TripleO::Services::Snmp
- OS::TripleO::Services::Sshd
- OS::TripleO::Services::SwiftProxy
- OS::TripleO::Services::SwiftDispersion
- OS::TripleO::Services::SwiftRingBuilder
- OS::TripleO::Services::SwiftStorage
- OS::TripleO::Services::Timesync
- OS::TripleO::Services::Timezone
- OS::TripleO::Services::TripleoFirewall
- OS::TripleO::Services::TripleoPackages
- OS::TripleO::Services::Tuned
- OS::TripleO::Services::Vpp
- OS::TripleO::Services::Zaqar

# Role: ComputeHCIovsDpdkSriov

description: |

ComputeOvsDpdkSriov Node role hosting Ceph OSD too

networks:
  InternalApi:
    subnet: internal_api_subnet
  Tenant:
    subnet: tenant_subnet
  Storage:
    subnet: storage_subnet
  StorageMgmt:
    subnet: storage_mgmt_subnet

# CephOSD present so serial has to be 1
update_serial: 1

RoleParametersDefault:
  TunedProfileName: "cpu-partitioning"
  VhostuserSocketGroup: "hugetlbfs"

NovaLibvirtRxQueueSize: 1024
NovaLibvirtTxQueueSize: 1024

ServicesDefault:
  - OS::TripleO::Services::Aide
  - OS::TripleO::Services::AuditD
  - OS::TripleO::Services::BootParams
  - OS::TripleO::Services::CACerts
  - OS::TripleO::Services::CephClient
  - OS::TripleO::Services::CephExternal
  - OS::TripleO::Services::CephOSD
  - OS::TripleO::Services::CertmongerUser
  - OS::TripleO::Services::Collectd
  - OS::TripleO::Services::ComputeCeilometerAgent
  - OS::TripleO::Services::ComputeNeutronCorePlugin
  - OS::TripleO::Services::ComputeNeutronL3Agent
  - OS::TripleO::Services::ComputeNeutronMetadataAgent
  - OS::TripleO::Services::ComputeNeutronOvsDpdk
  - OS::TripleO::Services::Docker
  - OS::TripleO::Services::IpaClient
  - OS::TripleO::Services::Ipsec
  - OS::TripleO::Services::Iscsid
  - OS::TripleO::Services::Kernel
  - OS::TripleO::Services::LoginDefs
  - OS::TripleO::Services::MetricsQdr
  - OS::TripleO::Services::Multipathd
  - OS::TripleO::Services::MySQLClient
  - OS::TripleO::Services::NeutronBgpVpnBagpipe
  - OS::TripleO::Services::NeutronSriovAgent
  - OS::TripleO::Services::NeutronSriovHostConfig
  - OS::TripleO::Services::NovaAZConfig
  - OS::TripleO::Services::NovaCompute
  - OS::TripleO::Services::NovaLibvirt
  - OS::TripleO::Services::NovaLibvirtGuests
  - OS::TripleO::Services::NovaMigrationTarget
  - OS::TripleO::Services::OvsDpdkNetcontrold
  - OS::TripleO::Services::ContainersLogrotateCrond
  - OS::TripleO::Services::Podman
  - OS::TripleO::Services::Rear
  - OS::TripleO::Services::Rhsm
  - OS::TripleO::Services::Rsyslog
  - OS::TripleO::Services::RsyslogSidecar
15.1.2. network-environment-overrides.yaml

---

parameter_defaults:

# The tunnel type for the tenant network (geneve or vlan). Set to "" to disable tunneling.
NeutronTunnelTypes: "geneve"

# The tenant network type for Neutron (vlan or geneve).
NeutronNetworkType: ["geneve", "vlan"]

# Nova flavor to use.
OvercloudControllerSriovFlavor: controller

# Number of nodes to deploy.
ControllerSriovCount: 3

# MTU global configuration
NeutronGlobalPhysnetMtu: 9000

# Configure the classname of the firewall driver to use for implementing security groups.
NeutronOVSFirewallDriver: openvswitch

# Enable log level DEBUG for supported components
Debug: true

# From Rocky live migration with NumaTopologyFilter disabled by default
NovaEnableNUMALiveMigration: true

NeutronPluginExtensions: "port_security,qos,segments,trunk,placement"

# Scheduler configuration
ControllerHostnameFormat: "controller-%index%"
ControllerSchedulerHints: "capabilities:node": "controller-%index%"
ComputeOvsDpdkSriovHostnameFormat: "computeovsdpksriov-%index%"
ComputeOvsDpdkSriovSchedulerHints: "capabilities:node": "compute-%index%"

# Enable log level DEBUG for supported components
Debug: true

NeutronServicePlugins: "ovn-router,trunk,qos,placement"

NeutronSriovAgentExtensions: "qos"
### NovaSchedulerEnabledFilters
- AvailabilityZoneFilter
- ComputeFilter
- ComputeCapabilitiesFilter
- ImagePropertiesFilter
- ServerGroupAntiAffinityFilter
- ServerGroupAffinityFilter
- PciPasssthroughFilter
- NUMATopologyFilter
- AggregateInstanceExtraSpecsFilter

```
NovaSchedulerEnabledFilters:
- AvailabilityZoneFilter
- ComputeFilter
- ComputeCapabilitiesFilter
- ImagePropertiesFilter
- ServerGroupAntiAffinityFilter
- ServerGroupAffinityFilter
- PciPasssthroughFilter
- NUMATopologyFilter
- AggregateInstanceExtraSpecsFilter
```

`ComputeOvsDpdkSriovNetworkConfigTemplate: "/home/stack/ospd-17.0-geneve-ovn-dpdk-sriov-ctiplane-dataplane-bonding-hybrid/nic-configs/computeovsdpkdsriov.yaml"

`ControllerSriovNetworkConfigTemplate: "/home/stack/ospd-17.0-geneve-ovn-dpdk-sriov-ctiplane-dataplane-bonding-hybrid/nic-configs/controller.yaml"

---

#### 15.1.3. controller.yaml

---

**network_config:**
- **type:** interface
  - **name:** nic1
  - **use_dhcp:** false
  - **addresses:**
    - **ip_netmask:** 
      
      {{ ctlplane_ip }}/{{ ctlplane_subnet_cidr }}
  - **routes:**
    - **ip_netmask:** 169.254.169.254/32
      - **next_hop:** 
        
        {{ ctlplane_ip }}

- **type:** linux_bond
  - **name:** bond_api
  - **bonding_options:** mode=active-backup
  - **use_dhcp:** false
  - **dns_servers:** 
    
    {{ ctlplane_dns_nameservers }}
  - **members:**
    - **type:** interface
      - **name:** nic2
        - **primary:** true

- **type:** vlan
  - **device:** bond_api
  - **vlan_id:** 
    
    {{ lookup('vars', networks_lower['InternalApi'] ~ '_vlan_id') }}
  - **addresses:**
    - **ip_netmask:** 
      
      {{ lookup('vars', networks_lower['InternalApi'] ~ '_ip') }}/{{ lookup('vars', networks_lower['InternalApi'] ~ '_cidr') }}

- **type:** vlan
  - **device:** bond_api
  - **vlan_id:** 
    
    {{ lookup('vars', networks_lower['Storage'] ~ '_vlan_id') }}
  - **addresses:**
    - **ip_netmask:** 
      
      {{ lookup('vars', networks_lower['Storage'] ~ '_ip') }}/{{ lookup('vars', networks_lower['Storage'] ~ '_cidr') }}

- **type:** vlan
  - **device:** bond_api
vlan_id: {{ lookup('vars', networks_lower['StorageMgmt'] ~ '_vlan_id') }}

addresses:
- ip_netmask: {{ lookup('vars', networks_lower['StorageMgmt'] ~ '_ip') }}/{{ lookup('vars', networks_lower['StorageMgmt'] ~ '_cidr') }}

- type: ovs_bridge
  name: br-link0
  use_dhcp: false
  mtu: 9000
  members:
  - type: interface
    name: nic3
    mtu: 9000

- type: vlan
  vlan_id: {{ lookup('vars', networks_lower['Tenant'] ~ '_vlan_id') }}
  mtu: 9000
  addresses:
  - ip_netmask: {{ lookup('vars', networks_lower['Tenant'] ~ '_ip') }}/{{ lookup('vars', networks_lower['Tenant'] ~ '_cidr') }}

- type: ovs_bridge
  name: br-dpdk0
  use_dhcp: false
  mtu: 9000
  members:
  - type: interface
    name: nic4
    mtu: 9000

- type: ovs_bridge
  name: br-dpdk1
  use_dhcp: false
  mtu: 9000
  members:
  - type: interface
    name: nic5
    mtu: 9000

- type: ovs_bridge
  name: br-sriov1
  use_dhcp: false
  mtu: 9000
  members:
  - type: interface
    name: nic6
    mtu: 9000

- type: ovs_bridge
  name: br-sriov2
  use_dhcp: false
  mtu: 9000
  members:
  - type: interface
    name: nic7
    mtu: 9000
- type: interface
  name: nic8
  use_dhcp: false
  defroute: false

- type: interface
  name: nic9
  use_dhcp: false
  defroute: false

- type: ovs_bridge
  name: br-access
  use_dhcp: false
  mtu: 9000
  members:
    - type: interface
      name: nic10
      mtu: 9000
    - type: vlan
      vlan_id: {{ lookup('vars', networks_lower['External'] ~ '_vlan_id') }}
      mtu: 9000
      addresses:
        - ip_netmask: {{ lookup('vars', networks_lower['External'] ~ '_ip') }}/{{ lookup('vars', networks_lower['External'] ~ '_cidr') }}
      routes:
        - default: true
          next_hop: {{ lookup('vars', networks_lower['External'] ~ '_gateway_ip') }}

outputs:
  config:
    value: get_attr[OsNetConfigImpl, value]

15.1.4. `compute-ovs-dpdk.yaml`

network_config:
- type: interface
  name: nic1
  use_dhcp: false
  default: no

- type: interface
  name: nic2
  use_dhcp: false
  addresses:
    - ip_netmask: {{ lookup('vars', networks_lower['External'] ~ '_ip') }}/{{ lookup('vars', networks_lower['External'] ~ '_cidr') }}
  routes:
    - ip_netmask: 169.254.169.254/32
      next_hop: {{ lookup('vars', networks_lower['External'] ~ '_gateway_ip') }}
    - default: true
      next_hop: {{ lookup('vars', networks_lower['External'] ~ '_gateway_ip') }}

- type: linux_bond
  name: bond_api
  use_dhcp: false
  bonding_options: "mode=active-backup"
dns_servers: {{ ctlplane_dns_nameservers }}
members:
- type: interface
  name: nic3
  primary: true
- type: interface
  name: nic4

- type: vlan
  vlan_id: {{ lookup('vars', networks_lower['InternalApi'] ~ '_vlan_id') }}
  device: bond_api
  addresses:
  - ip_netmask: {{ lookup('vars', networks_lower['InternalApi'] ~ '_ip') }}/{{ lookup('vars', networks_lower['InternalApi'] ~ '_cidr') }}

- type: vlan
  vlan_id: {{ lookup('vars', networks_lower['Storage'] ~ '_vlan_id') }}
  device: bond_api
  addresses:
  - ip_netmask: {{ lookup('vars', networks_lower['Storage'] ~ '_ip') }}/{{ lookup('vars', networks_lower['Storage'] ~ '_cidr') }}

- type: ovs_user_bridge
  name: br-link0
  use_dhcp: false
  ovs_extra: "set port br-link0 tag={{ lookup('vars', networks_lower['Tenant'] ~ '_vlan_id') }}"
  addresses:
  - ip_netmask: {{ lookup('vars', networks_lower['Tenant'] ~ '_ip') }}/{{ lookup('vars', networks_lower['Tenant'] ~ '_cidr') }}

members:
- type: ovs_dpdk_bond
  name: dpdkbond0
  rx_queue: 1
  ovs_extra: "set port dpdkbond0 bond_mode=balance-slb"
members:
- type: ovs_dpdk_port
  name: dpdk0
  members:
  - type: interface
    name: nic7
  - type: ovs_dpdk_port
    name: dpdk1
    members:
    - type: interface
      name: nic8

- type: ovs_user_bridge
  name: br-dpdk0
  use_dhcp: false
  mtu: 9000
  rx_queue: 1
members:
- type: ovs_dpdk_port
  name: dpdk2
  members:
  - type: interface
name: nic5
- type: ovs_user_bridge
  name: br-dpdk1
  use_dhcp: false
  mtu: 9000
  rx_queue: 1
  members:
    - type: ovs_dpdk_port
      name: dpdk3
      members:
        - type: interface
          name: nic6
    - type: sriov_pf
      name: nic9
      mtu: 9000
      numvfs: 10
      use_dhcp: false
      defroute: false
      nm_controlled: true
      hotplug: true
      promisc: false
- type: sriov_pf
  name: nic10
  mtu: 9000
  numvfs: 10
  use_dhcp: false
  defroute: false
  nm_controlled: true
  hotplug: true
  promisc: false

outputs:
  config:
    value: get_attr[OsNetConfigImpl, value]

15.1.5. overcloud_deploy.sh

#!/bin/bash

tht_path=/home/stack/ospd-17.0-geneve-ovn-dpdk-sriov-ctlplane-dataplane-bonding-hybrid
[[ ! -d "$tht_pathroles" ]] && mkdir $tht_path/roles
openstack overcloud roles generate -o $tht_path/roles/roles_data.yaml ControllerSriov ComputeOvsDpkSriov
openstack overcloud deploy \
  --templates /usr/share/openstack-tripleo-heat-templates \
  --ntp-server clock.redhat.com,time1.google.com,time2.google.com,time3.google.com,time4.google.com \
  --stack overcloud \
  --roles-file $tht_path/roles/roles_data.yaml \
  -n $tht_path/network/network_data_v2.yaml \
  --deployed-server \

Red Hat OpenStack Platform 17.1-Beta Configuring network functions virtualization

```
-e /home/stack/templates/overcloud-baremetal-deployed.yaml \ 
-e /home/stack/templates/overcloud-networks-deployed.yaml \ 
-e /home/stack/templates/overcloud-vip-deployed.yaml \ 
-e /usr/share/openstack-tripleo-heat-templates/environments/services/neutron-ovn-ha.yaml \ 
-e /usr/share/openstack-tripleo-heat-templates/environments/services/neutron-ovn-dpdk.yaml \ 
-e /usr/share/openstack-tripleo-heat-templates/environments/services/neutron-ovn-sriov.yaml \ 
-e /home/stack/containers-prepare-parameter.yaml \ 
-e $tht_path/network-environment-overrides.yaml \ 
-e $tht_path/api-policies.yaml \ 
-e $tht_path/bridge-mappings.yaml \ 
-e $tht_path/neutron-vlan-ranges.yaml \ 
-e $tht_path/dpdk-config.yaml \ 
-e $tht_path/sriov-config.yaml \ 
--log-file overcloud_deployment.log
```